Improving Performance and Security in Anonymity Systems

by

MOHSEN IMANI

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2018

To *my parents*.

ACKNOWLEDGEMENTS

I would like to acknowledge and express my deepest appreciation to my supervising professor Prof. Matthew Wright for the patient guidance, encouragement and advice he has provided throughout my PhD study. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries promptly. The research in this dissertation would not have been possible without his constant support and mentoring. He had introduced me to a diverse set of fascinating problems and enabled collaboration with numerous great researchers. His teaching, mentoring, managing, and research had a huge positive impact on me. I would also like to thank Prof. Gergely Zaruba who helped me in my supervisors absence.

I would like to thank the rest of my committee: Prof. Manfred Huber, Prof. Farhad Kamangar for their encouragement, and insightful comments.

I am grateful to my research collaborators who have introduced me to a number of interesting problems. I have learned a lot from these incredible researchers by observing their style of research. Marc Juarez and Claudia Diaz of KU Leuven, Payap Sirinam of Rochester Institute of Technology, and, Armon Barton of the University of Texas at Arlington had been my consistent source of inspiration and encouragement. I have learned an incredible amount from them. Their intellect and hard work have always been an inspiration to me.

I would also like to extend my gratitude to the department of CSE at UTA and Prof. Wright for providing me with financial support during my graduate studies. I also like to thank my lab mates Jees, Taiabul, John, Revanth, Mehrdad, Mahdi, Max, and Omid. Many

thanks to my friends Vahidreza, Majid, Hamidreza, Bahram, Hussein, Pooria, Peyman, Abolfazl, and Hooman for making my PhD memorable.

Finally, I would like to convey my heartfelt gratitude to my parents, brother, sisters and my vast extended family. Their constant encouragement and support have been critical to my success.

August 2018

ABSTRACT

Improving Performance and Security in Anonymity Systems

Mohsen Imani, Ph.D.

The University of Texas at Arlington, 2018

Supervising Professor: Gergely Zaruba

Tor is an anonymity network that provides online privacy for the Internet users. Tor hides the user's traffic among the others' traffic. The more users Tor attracts, the stronger anonymity it provides. Unfortunately, users of the Tor anonymity system suffer from less-than-ideal performance, in part because circuit building and selection processes are not tuned for speed. Moreover, there are some attacks like *guard fingerprinting* and *website fingerprinting* attacks that try to profile or de-anonymize the Tor users. In this dissertation, we propose methods to address both security and performance issues in Tor. We first examine the process of selecting among pre-built circuits and the process of selecting the path of relays for use in building new circuits to improve performance while maintaining anonymity. We also propose a method to improve the mechanism of picking guards in Tor. The guard selection mechanism in Tor suffers from security problems like *guard fingerprinting* and from performance issues. To address this problem, we propose a new method for forming guard sets based on Internet location. We construct a hierarchy that keeps clients and guards together more reliably and prevents guards from easily joining arbitrary guard sets. This approach also has the advantage of confining an attacker with access to

limited locations on the Internet to a small number of guard sets. Tor is also known to be vulnerable to the traffic analysis attacks like *Website Fingerprinting* (WF) attacks. In WF attacks, the adversary attempts to identify the websites visited by the user. We also propose a method using adversarial examples to decrease the accuracy rate of the WF attack. We generate adversarial traces to cause misclassification in the WF attackers. We show that if the WF attacker trains its classifier on the adversarial traces, they are not effective WF defenses. We propose a method to solve this problem, and we show that our method can drop the WF attacker's accuracy from 98% to 60% with 47% bandwidth overhead.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

Introduction

Privacy is a human right, the right to be alone and free from intrusion and interference, and it maintains the human dignity and respect. The information privacy is about the control on how our data and information is used, collected, stored, and shared. By the advance of the technology, the information privacy gets more and more complicated, in a way that any bit of our information is getting valuable. Companies and intelligence agencies collect tremendous amount of data about us, sometimes without our consent. Based on that data, they draw conclusion about us that we may disagree with them.

Security and privacy are cousins. Privacy is about the choice and having power on our data. Security protects our data from the malicious hands. Authentication, authorization, and encryption are the security mechanisms protecting the access level and the content of the data. These mechanisms are actually necessary for security and protecting the data but not sufficient for addressing the privacy. For example, in TLS-enabled applications, although the data is encrypted and the man in the middle cannot read the content, he can still see who is talking to whom.

Sometimes, we want people to know about something or what we do but we want to be unknown, this part of privacy is anonymity. People choose to be anonymous for different situations, such as charity, witness of a crime, participating in a crime, and etc. The Internet does not provide anonymity by design. Our IP address in the Internet is like our mailing address that releases our identity. Our IP address can be traced back to our real identity. Our IP address is a part of an address space of the ISP (Internet Service Provider) serving

us. The ISPs have information about their customers and it can release the real identity behind an IP address.

Fortunately, there are some privacy enhancing Technologies (PETs) that build an infrastructure over the Internet and add anonymous communication feature to the Internet. The simplest tools to achieve anonymity are technologies that use single proxies. In these technologies, the clients do not directly contact their destinations, they first contact a proxy and ask the proxy to contact the intended destination. Therefore, the network observer see the client contacts the proxy but they do not know what the destination is. These technologies consider a weak network observer and put all the trust on a single point, the proxy. The proxy should be trusted because it knows both ends of the traffic. Even if the proxy is trusted, a network observer watching ingress and egress in the proxy can correlate the traffic to identify who is talking to whom and break the anonymity. The next generation of PETs, such as Tor network [2], Mix networks [3], and I2P [4], have the distributed structures that distribute the trust over multiple points in the network. The clients traffic in these technologies travels through multiple random nodes in the network. No individual node knows the complete path. Thus, the attacker needs to be on the entry and exit points of the traffic to be able to link the client and destination by the traffic correlation. Among the existing anonymity networks, we mainly focus on Tor because it is the most popular low-latency anonymity network.

Tor is a volunteer-based overlay network that thousands privacy advocates donate their bandwidth to keep Tor running and functional. Tor passes the user's traffic through three almost-randomly selected proxies that no single point in the network can link the user to her destination. Tor is a low–latency anonymity network, which means it does not modify the inter-packet delay. Tor has created a bunch of interesting services, such as web browsing, hidden services, Tor mail, and Tor chat. These services attract almost 2 millions daily users for Tor.

1.1 Motivations

Anonymity networks like Tor work by hiding the users among the users. The larger number of users, the more anonymous the users become. Clients in Tor are both consumers of anonymity and, as a part of the system, the providers of cover traffic for the others' anonymity. Therefore, motivating more Internet users to use Tor in their on-line life strengthens the anonymity provided by Tor and increases the *anonymity set*. The anonymity set is the number of people whom the attacker can guess to be the one involved in the some activity of interest [5]. Usability, which is related to the users' experience with the system, contributes to the security. In the case of Tor, better usability and user experience will attract more and more users and makes the provided anonymity stronger (larger anonymity sets).

The usability is not only about the user experience with Tor software, the more damaging issue to the Tor usability is *performance*. Unfortunately, Tor suffers from less-than-optimal performance which can discourages some of the Internet users to use Tor [6]. Improving the performance not only results in better user experience, but also improves the security. Therefore, addressing the performance issue is an importance matter. In this dissertation, we address this problem and propose some methods to improve the Tor performance.

On the other hand, several attacks [7, 8, 9, 10, 11, 12, 13], have been introduced that threaten the users' anonymity. The existence of these attacks and emergence of new attacks may discourage the Internet users to use Tor. This even deteriorates the damage of the attacks because dropping the number of users causes the small anonymity sets and weakens the anonymity. Therefore, designing some sort of defenses against these attacks is crucial in Tor. As the next step in this dissertation, we address this problem and propose some defenses against attacks trying to de-anonymize the Tor users.

1.2 Contributions

In this dissertation we address both performance and security issues in Tor and make the following fundamental contributions:

- **Improving the performance through the *circuit* selection** (*Chapter 3*). The Tor client's traffic traverses in the network through the *circuits*. Circuits are the encrypted connection of (three) relays on the network. The Tor client does not consider any performance metrics in the selection of the circuits to be used for the traffic. Without modifying the way the circuits are built in the Tor client, we propose several performance metrics to select the fast circuits. We found that selecting the circuits based on their *round trip time* (*RTT*) offers high performance (20% reduction in the network responsiveness compared to the state-of-the-art method) without losing the anonymity.
- **Improving the performance through the relay selection** (*Chapter 3*). Circuits in Tor are the encrypted connections between three relays. The selection of the relays in the circuit creation is biased toward the bandwidth. The high bandwidth relays have higher selection probabilities. The relay selection based on bandwidth causes to build circuits that have high bandwidth relays but there may be multiple intercontinental hopes due to the relays' location diversity. We propose a method that considers the location of the relays along with their bandwidth in the relay selection to avoid long circuits.
- **Designing a defense against the *guard fingerprinting* attacks** (*Chapter 4*). The first relay in the circuit is in a very critical position, *guard position*, because it is in the direct contact with the client. Unfortunately, not all the relays have the bandwidth and uptime capability to be selected as the first hop. The number of the relays that are eligible to be selected in the first hop are limited and the number of clients are much larger than guards, which causes small anonymity sets (the number of clients using the same guard). Therefore, the guard can act as the fingerprint of the client that can lead to *guard*

fingerprinting attack [14]. We propose a method that groups the guards into few sets and the clients select one of these sets to pick their guards from.

- **Designing a defense against the *website fingerprinting* attacks** (*Chapter 5*). Tor network does not change the inter-packet timings and the order of the packets. An adversary located somewhere in the first hop (between the client and the first relay of the circuit, i.e. guard) can take advantage of this information to identify the client's web browsing activities. This attack is called *website fingerprinting* attack [15, 16, 17, 18, 19, 20]. To lower the accuracy of this attack, we propose to leverage the *adversarial examples* concept in the machine learning field [21, 22, 23, 24] to reshape the traffic and cause mis-identification in the attacker.

1.3 Dissertation Outline

We start with a background on Tor and how it works in Chapter 2. Then we address both security and performance problem in Tor and introduce methods to improve them. In Chapter 3, we improve the performance by modifying the circuit selection mechanism and relay selection mechanism in Tor. We improve the performance by selecting the circuits (at the time of stream attachment) based on some performance metrics instead of picking them randomly. We also modify the relay selection and engage the location of the relays in the selection as well as their bandwidth. In Chapter 4, we propose a guard selection mechanism to improve the security of Tor against a certain class of attacks like *guard fingerprinting* attacks. In Chapter 5, we introduce a defense mechanism using the adversarial machine learning to prevent the *website fingerprinting* attacks.

CHAPTER 2

Background

The Tor network consists of around 7,000 relays, called Onion Routers (ORs). These ORs are run by volunteers which donate a portion of their bandwidth to the Tor network. Therefore, the number of relays and the relays contributing to the Tor network change over time due to the leaving and joining of ORs. ORs provide information about their donated bandwidth, IP addresses and ports, and *exit policies*—the addresses and ports they are willing to be connected to external Internet destinations—to a small group of servers called *directory authorities*. The directory authorities are trusted relays that information about them is hard coded in the Tor source code. The directory authorities assign flags to the ORs based on their availability, bandwidth and exit policies. Then, they mutually agree upon a list of all information about the ORs in the network, this list is called as the *consensus document*.

A Tor client, called the *Onion Proxy* (OP), first contacts one of directory authorities or their mirrors and downloads the consensus to get the current status of the Tor network. Since the Tor network is dynamic, with relays regularly joining and leaving, the directory authorities update the consensus hourly. The OP uses the consensus information to select a path of three relays to use in communicating with its destinations.

The OP first picks an *exit* node, the last OR in the path and the one responsible for communicating directly with the user's intended destinations (e.g. web servers). Only a subset of the ORs allows exit traffic, and the directory authorities mark these ORs with the *Exit* flag in the consensus. The exit node in the path is selected among the ORs with Exit flag in the consensus. The second OR to be selected is the *entry* node or the guard node, the

one that the OP will communicate with directly. The entry node is selected among the ORs with *Guard* flag in the consensus. The directory authorities flag the high uptime and high bandwidth ORs as Guards. Finally, the OP picks a *middle* node from all the ORs. The OP selects these nodes (exit node, entry node, and middle node) randomly with a bias towards higher bandwidth relays for load balancing and performance reasons. Additionally, no two ORs from the same /16 subnet or who are controlled by the same group of relay owners should appear on the same path [2].

Once the OP picks this path of ORs, it then sets out to build a *circuit* of layered cryptographic connections through this path. To prevent the fingerprint attack through the packet sizes, introduced by [20], the user' traffic goes through the circuit in fixed-size cells of 512 bytes. The circuit is created by the following steps:

- The OP sends a *CREATE* cell to the guard relay. The CREATE cell includes a unique Circuit ID, and the first half of a Diffie-Hellman handshake g^{x_1}. The cell is encrypted with the guard nodes public key extracted from the descriptors file downloaded from the directory authorities.
- The guard relay responds back to the OP handshake with the *CREATED* cell. The cell contains the second half of the Diffie-Hellman handshake g^{y_1} along with a hash of the negotiated key $K_1 = g^{y_1 x_1}$. Symmetric key K_1 is used for communicating between the OP and guard.
- The OP sends an *EXTEND* cell to the guard node. This cell contains the information about the middle, and encrypted g^{x_2} (with the middle node's public key).
- The guard node copies the encrypted g^{x_2} into a CREATE cell and sends it to the middle node.
- The middle node responds back to the guard node with a CREATED cell. This cell contains g^{y_2} along with a hash of the negotiated key $K_2 = g^{y_2 x_2}$.

- The guard node wraps the payload of the CREATED cell into a *EXTENDED* cell and passes it back to the OP. The circuit is now extended to the middle node.
- The OP sends an *EXTEND* cell to the middle node containing the information about the exit node, and encrypted g^{x_3} (with exit node's public key). This cell is encrypted with key K_2 and passes through the guard node. The guard node does not know about the content of the cell.
- The middle node copies the encrypted g^{x_3} into a CREATE cell and sends it to the exit node.
- The exit node responds back to middle node with CREATED cell. This cell contains g^{y_3} along with a hash of the negotiated key $K_3 = g^{y_3 x_3}$.
- The middle node wraps the payload of the CREATED cell into a *EXTENDED* cell and passes it back to the guard node. The guard adds one more layer of encryption with key K_1 and passes it to the OP.

After this process the secret keys (K_1, K_2, and K_3) have been shared between OP and each node on the path. Once the OP sends the cell down the circuit, it encrypts the cell three times with key K_3, K_2, K_1, respectively. As the cell traverses the circuit, the cell is decrypted one layer at time. Once the circuit reaches the exit node, it has only one layer of encryption (with K_3) and the exit decrypts that layer. The traffic between the exit and the destination is not encrypted by the Tor network.

When the user's application makes a request through Tor, the Tor client will first check to see if it has an open circuit available to attach the stream to it. Typically, the client maintains one to three open circuits, as building a circuit takes time that would further slow down the user's experience. In particular, the client checks once per second to see whether there are at least two open circuits and creates new circuits if needed. Circuits that have been used for 10 minutes are marked as *dirty*, and they are not used for future connections, which means that a new circuit will be needed. More circuits can also be added if the

user's application requires ports that are not allowed on the exit policies of the currently open circuits. Additionally, the Tor client maintains circuits for hidden services, which are servers that can only be accessed through the Tor network to protect the privacy of not only the user but the service itself, and one-hop circuits (to entries only) that are used to download the consensus. Geddes et al. report that the Tor client maintains an average of 10 circuits [25], though we note that usually only two of these are available for web browsing.

2.0.1 Onion Services

Tor not only helps the users to hide their location and their identity, but also helps them to offer various kinds of services, such as web publishing, chat services, and email. These services are called Onion Services, (OSs). The OSs' location and identity are hidden through Tor network. In the following paragraphs, we explain the Tor onion service protocol.

Server Setup: The onion server randomly picks 3 to 5 Tor relays and lets them know about its *public key*. These relays are called *Introduction Points* (IPs). The *Introduction Point* is the middle point between the client and the onion server that lets the onion server knows someone is interested in its service.

The onion server needs to announces its *Introduction Points* to the network. The onion server generates a document called *onion service descriptor* and uploads it to a distributed hash table (DHT). The descriptor contains the server's public key and a summary of each introduction point, and it signed it by its private key.

Client Connection The clients find the address of the onion server through the out-of-band channels. The address of the onion server is a 16-character name derived from the server's public key. The client uses the OS address to look for the relays that hold the OS's descriptor in the DHT to download the descriptor.

The client randomly picks a relay as the *Rendezvous Point* (RP), creates a three-hop circuit to RP, and gives the RP a one-time secret (encrypted by the OS's public key). The RP does not know about the client's location because the communication between RP and the client is through a three-hop circuit.

The OS's descriptor contains the *Introduction Points*. The client contacts one of the Introduction Points through a three-hop circuit and asks the *Introduction Point* to pass an *Introduce* message (encrypted by the OS's public key) to the OS. The *Introduce* message includes the address of the RP and the one-time secret.

Final Connection Once the OS receives the *Introduce* message, it decrypts the message and finds out the address of RP and the one-time secret. The OS creates a circuit to the RP and gives it the secret. The RP connects two ends of the two circuits, the client's circuit to the RP and the OS's circuit to the RP.

CHAPTER 3

Performance Improvement

3.1 Introduction

Tor provides anonymity for millions of users around the world by routing their traffic over paths selected from approximately 7,000 volunteer-run relays.[1] Tor effectively hides the user among all the users, so having more users and more traffic enhances anonymity for all [26, 5]. Unfortunately, Tor users often face large delays and long download times, which can discourage users and thereby reduce anonymity. In this paper, we examine two approaches to improve Tor performance and evaluate them in term of both performance and security.

Circuit Selection. The client's traffic in Tor goes through a three-hop encrypted channel, called a *circuit*. When the user makes a request, such as for a webpage, Tor attaches the new stream (by opening a SOCKS connection) to a circuit. The Tor client builds circuits preemptively based on the client's use or immediately if there is no current circuit to handle the stream. Tor currently does not use any performance criteria in selecting a circuit. In this paper, we evaluate using the length of the circuits, their congestion, the Round Trip Time (RTT), or a combination of them in choosing a fast circuit. We also find that the number of available circuits in Tor is often small, between one and three circuits, such that picking the best circuit for performance does not have much effect in practice. As the number of available circuits increases, the chance of finding a fast and high performance circuit should increase. To this end, for each circuit selection criteria we study, we evaluate the impact of more available circuits in terms of both performance and security.

[1] `https://metrics.torproject.org/`, accessed August. 2016

Relay Selection. For circuit selection to be effective, some of the available circuits must be reasonably high performing. To improve the chances of this, we modify the relay selection mechanism to build short and high-bandwidth circuits. Tor clients select paths in a way that balances traffic load among the relays according to their advertised bandwidths, but they do not make any consideration for the locations of relays relative to the clients, their destinations, or the other relays in the path. Paths can jump around the globe, which is *intuitively* good for anonymity but *measurably* bad for performance.

Prior work has examined improving path selection in Tor for better performance, considering factors such as bandwidth [27], congestion [28], latency [29], and location [30].

Wacek et al. performed a comprehensive study of path selection [31], and they found that congestion-aware routing [28] offers the best combination of performance and anonymity among the tested approaches. They also found that approaches that emphasized latency but failed to consider bandwidth had poor performance, and they suggested that an approach that optimized both latency and bandwidth could do better than any of their tested approaches. In this paper, we take on this suggestion and explore designs that address both criteria.

We make the following contributions:

- We define nine circuit selection approaches using the geographical length, circuit delay, congestion, or a combination of these. We evaluate each of the approaches and compare them experimentally.
- In our relay selection approach, *combined weighting*, we explore the design of a single weighting function that balances bandwidth and geographical inter-node distance. We examine the design issues in our approach and compare it with the state of the art.
- To prevent delays, it is important to build circuits in advance of their use [31]. Since we want to use destination location information to better inform our path-selection strategy,

we build circuits in advance, using popular destinations as the end points. We then consider the circuits' RTTs, base on our findings in circuit selection approaches, to select from among these circuits.

- We show the results of experiments on our approaches in Shadow, following the methodology of Jansen et al. [32], and we examine a range of parameters. We find a number of settings in our proposed methods that offer reasonable anonymity and significant performance improvements over congestion-aware routing, the current state-of-the-art in Tor path selection. In particular, our recommended approach provides a 20% reduction in median time to first byte and a 11% reduction in median time to last byte compared to congestion-aware routing.
- We also measure the security of our approaches using Gini coefficient and entropy on first-and-last combinations, with the rate of path compromise in the presence of relay-level and AS-level adversaries, and in the presence of four targeted relay-level attacks. We find that both of our approaches provide anonymity in line with Tor at settings that also provide significant performance improvements. Our recommended approach has a slightly better Gini coefficient and entropy than Tor, with slightly fewer compromised paths against our attackers.

3.2 Related work

Researchers have addressed Tor performance issues in a variety of ways, such as modifying circuit scheduling [33], congestion control [28], traffic splitting [34], and incentives to encourage users to offer their bandwidth [35, 36]. In this section, we first briefly overview Tor's current path selection mechanism and then discuss the prior works on enhancing performance in Tor from circuit selection and relay selection point of view.

Tor. Tor is a volunteer-based overlay network providing anonymity online. Details are available at `http://www.torproject.org/` and in the original design paper [37]. In Tor, the choice of relays is governed a complex weighting function[2] that includes various considerations and the bandwidths of the relays. Weighting by bandwidths serves to balance load in the system, as relays have huge variance in advertised bandwidth, with the bottom quintile under 2 Mbps, a median of about 10 Mbps, and a maximum of 1 Gbps as of May 2016.[3]

Circuit Selection. Can et al. [33] propose a circuit scheduling mechanism that gives high priority to interactive traffic over bulk traffic on the same connection. This circuit selection mechanism has been deployed in Tor relays, but it has no impact on the client. Our circuit selection approaches are designed to improve the performance from the client side and are thus orthogonal to scheduling in the relays.

Wang et al. introduce *node latency* as a parameter to measure a relay's congestion [28]. In their approach, *congestion-aware routing (CAR)*, the client calculates congestion delay using both active and opportunistic methods. It then uses the measured latency to avoid congested nodes during path selection and to avoid selecting congested paths. They use both *short-term* and *long-term* congestion in their work, where short-term congestion is caused by current traffic levels and long-term congestion is caused by the relay's bandwidth. Their results show improvement in quality of service and load balancing. In our evaluation of circuit selection methods, we also examine the use of congestion times and compare them with RTTs and circuit lengths.

The current Tor client measures the Circuit Build Time (CBT), i.e. the time to construct the circuit, and uses this to discard slow circuits whose CBT is above a client-specific threshold. Annessi and Schmiedecker [38] propose that Tor should use the circuit round

[2]Full details at `https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt`.
[3]`http://metrics.torproject.org/`

trip times (RTTs) in eliminating slow circuits instead of CBTs. In this method, the circuit RTT is actively measured after the circuit is built, and if it is longer than a timeout the circuit is discarded from future uses. In their study, this provided only 3% improvement in the time to download the first byte with mixed anonymity results. Our strategy in this paper is different from both approaches. Rather than examining circuits after their creation to discard them or keep them, we instead try to pick a high performing circuits in the first place.

Relay Selection. A number of improvements to Tor path selection have been investigated [27, 29, 30]. Wacek et al. examine Tor path selection in a comprehensive study with experiments running many simultaneous clients [31]. They create a model of the Tor network to evaluate the recent published papers modifying path selection and show results for throughput, time to last byte (TTLB), and round-trip time (RTT). They tested Tor, Snader/Borisov [27], Unweighted Tor, in which Tor relays get selected uniformly at random, Coordinate [29] in which path selection is based on estimated pair-wise latencies, LASTor [30], and Congestion-Aware [28]. Their investigation shows that path selection algorithms that do not consider bandwidth as a factor in relay selection have poor performance. Congestion-aware had nearly the best performance in throughput and time-to-first-byte, plus it had anonymity approximately in line with Tor and significantly better than other high-performing algorithms. We thus select it for comparison in our work.

Improving performance can also affect attackers, potentially providing them better attacking opportunities and more accurate measurements. There are several attacks that use latency and throughput information to de-anonymize Tor users [10, 11, 39]. Geddes et al. [40] introduced a new class of attacks, called *induced throttling*, that exploit performance-enhancing mechanisms to throttle and unthrottle a circuit and identify the user. They evaluated the vulnerability of performance improvements, such as congestion

control and traffic admission control to these attacks, and they found that there are highly effective attacks that can uniquely identify users. While this does not directly affect our approaches, we recognize that there is generally a trade-off between anonymity and performance.

3.3 Model and Goals

3.3.1 Network Model

Testing new path and circuit selection strategies on the live Tor network is challenging and could compromise users' anonymity or their harm their performance. We perform our simulations in Shadow [41, 42], a discrete-event simulator that runs the Tor code in a complete, but scaled-down, network. Shadow simulates the underlying network and it considers network attributes such as packet loss, bandwidth upstream and downstream, jitter, latency, and network edges. In our performance evaluations, we used a scaled-down model of Tor, which consists of 1100 clients and 220 relays; this scaled-down model was built based on the procedures of Jansen et al. [32] and measurements from the live Tor network (from July 2015). In our security evaluations, we use a larger scaled-down model of 2127 relays with one client at a time.

3.3.2 Attacker Model

As with prior work in Tor performance [27, 29, 30, 31], our attention is more on performance characteristics than on attacks. We only seek to validate that our approach does not significantly weaken the anonymity provided by Tor currently. We evaluate the security of our proposed mechanisms in terms of both relay-level and network-level adversaries.

Relay-Level Adversary Model. In the relay-level adversary model, we assume that the adversary is running some Tor relays in the network with the goal of getting into the guard

and exit positions of some circuits. An adversary in such a position can observe the entry and exit traffic and correlate them to link the clients to their destinations. To evaluate the security of our proposed circuit selection mechanism, we simulate our proposed method, CAR, and *vanilla* Tor in Shadow and randomly mark one of our guards and one of our exits as malicious relays. We then extract the streams and identify which ones were compromised. We repeat this process 10 times, and measure the compromise rates all over 10 repetitions.

To evaluate the security of our proposed relay selection mechanism at the relay level, we first follow the approach of Wacek et al., who use the Gini coefficient and entropy as measures of the diversity of paths taken by each of their studied approaches [31]. We consider a high-bandwidth attacker who adds a modest number of high-bandwidth ORs into the Tor network. Since our path selection algorithm uses distance as well as bandwidth, leading to our path selection algorithms pick high-bandwidth ORs with short distance more often, this attacker is aimed at capturing a large number of circuits. We also consider four targeted attack strategies in which the attacker targets a specific client, a specific destination, a specific client and destination, or with no specific target. In all these strategies, the attacker places his relays in the target's exact location to have minimum distance and a high chance to be selected. Our targeted attack scenarios are thus worst cases.

Network-Level Adversary Model. The adversary can control some network components like ASes or IPXs. If the entry traffic and exit traffic of an anonymous connection traverse through the adversary's network components, the adversary observes both sides of the traffic and deanonymizes the clients. We evaluate the security of our circuits selection mechanisms and relays selection mechanisms in the network level. In circuits selection and relay selection mechanisms, we simulate each of the proposed mechanisms in Shadow and extract the streams, including their paths. To determine the compromised streams, we use

the algorithm proposed by Qiu and Gao [43] to infer the AS paths on both the entry side of the circuit (between clients and guards) and and exit side (between exits and servers). Qiu and Gao's algorithm exploits known paths from BGP tables to improve the inferred paths. In measuring the compromise rates, we consider the possibility of an asymmetric traffic correlation attack that can happen between *data* path and *ack* path, which is one of the RAPTOR attacks proposed by Sun et al. [44].

3.3.3 Design Goals

We seek an algorithm that meets the following goals:

1. Interactive use like web browsing should be significantly faster than Tor and prior work.
2. Performance for bulk downloads should not be significantly slowed compared to Tor.
3. Anonymity should be similar to what Tor currently provides against our selected attack models.
4. Usage should be fairly distributed among relays according to their available capacities.
5. Clients should be able to select paths with little computational or other overhead.
6. Circuits should be available to the client for attaching streams to when needed.
7. We should avoid downloading large amounts of additional information from the directory servers.

We emphasize web traffic since delays in interactive use are more harmful to the user experience than delays in bulk downloads. We consider both response time, measured as time to first byte (TTFB), and total download time, measured as time to last byte (TTLB).

Note that we do not seek the optimal latency for circuits. Although having accurate latency information instead of geographic distance could further improve performance, the gains might be marginal given requirements for bandwidth and path diversity. Further,

obtaining and distributing accurate pairwise latency information may be expensive due to the necessary measurements and directory server overhead.

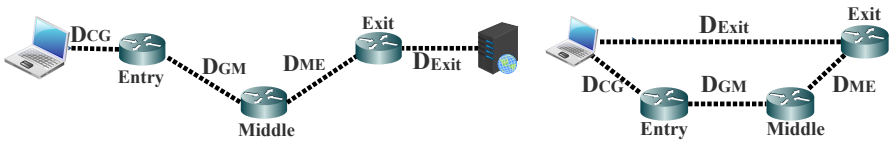
3.4 Circuit Selection

When a Tor client issues a request, the new stream is handled by one of the available circuits. In this section we explain how Tor tries to provide some available circuits for new streams and how it attaches the streams to the circuits. Then we explain how the stream attachment can be improved by increasing the number of circuits and considering performance criteria in circuit selection.

Pre-Built Circuits. As the user browses the Web with Tor, the Tor client opens new circuits so that later streams can be attached to those circuits without delay. Since different exits support different sets of ports, the Tor client aims to keep open two circuits to cover any port that the user has used recently. In practice, one or two circuits are typically available at any given time.

On-Demand Circuits. Sometimes the user's requested streams are not supported by current available circuits, or all available circuits are older than 10 minutes and considered *dirty*. In this case, Tor builds a circuit for the unhandled stream and attaches the stream to this circuit. It is obvious that these streams experience more delay than streams using pre-built circuits due to the circuit built time.

Tor Stream Attachment. When a new stream is created, the Tor client selects the most recently created circuit or creates a new circuit if needed and attaches the new stream to it. Then all communication on that stream, including DNS resolution, goes through the circuit.



(a) When the dest. IP is known (b) When the dest. IP is unknown

Figure 3.1. Distances used to compute circuit length..

3.4.1 Performance in Circuit Selection

The Tor client does not use performance as a criteria when selecting from available circuits for attaching a stream. Wang et al. [28] propose to use the least congested circuit, but there are several possible performance characteristics to use instead. The number of available circuits are often small, such that existing circuit selection mechanisms are not effective in practice. Also, we know of no study testing the effect of changing number of circuits on performance-based selection.

To investigate the effect of circuit selection on Tor performance, we evaluate both the number of available circuits for the streams and the way to choose the best circuit among the available circuits. To set the number of circuits, we check once per second that there are at least N circuits to support all recently used ports. If there are fewer than N circuits, then we start building circuits to reach the threshold. We compare Vanilla Tor, which typically offers one or two circuits, with making at least $N = 3$ to $N = 5$ circuits available at all times. Given some number of circuits, we can then use various methods to select the best one. We compare various combinations of geographic circuit length, congestion, and round trip time (RTT).

Metrics. The three basic metrics that we use are geographic circuit length, congestion time, and round-trip time. To find the total geographic circuit length (or simply *length*) L between the client and destination, we compute:

$$L = D_{CG} + D_{GM} + D_{ME} + D_{Exit} \tag{3.1}$$

D_{CG}, D_{GM}, D_{ME}, and D_{Exit} are shown in Figure 3.1(a) for when the destination IP address is known and in Figure 3.1(b) for when the IP address is not yet known. We use the opportunistic circuit measurements and latency model proposed by Wang et al. [28] to measure the circuit round-trip times and congestion times. Congestion time T_c is measured as:

$$T_c = RTT - RTT_{min} \tag{3.2}$$

where RTT is the round-trip time and RTT_{min} is the minimum RTT observed over that circuit. Wang et al. [28] showed that five measurements can effectively identify congested circuits. We thus measure and store the mean of the last five T_c measurements as the congestion time of the circuit, and the mean of the last five RTTs as the circuit RTT.

3.4.2 Attaching Streams to Circuits

We consider nine different methods in handling streams using circuit length, congestion, and RTT.

1. *Congestion Only*: Pick the circuit with the lowest congestion time.
2. *Length Only*: Pick the shortest circuit.
3. *RTT Only*: Pick the circuit with the lowest RTT.
4. *Congestion then length*: Select the two lowest congestion times and pick the shorter circuit.
5. *RTT then length*: Select the two lowest RTTs and pick the shorter one.

6. *Length then congestion*: Select the two shortest circuits and pick the lower congestion time.
7. *Length then RTT*: Select the two shortest circuits and pick the lower RTT.
8. *RTT then Congestion*: Select the two circuits with the lowest RTTs and pick the lower congestion time.
9. *Congestion then RTT*: Select the two circuits with lowest congestion times and pick the lower RTT.

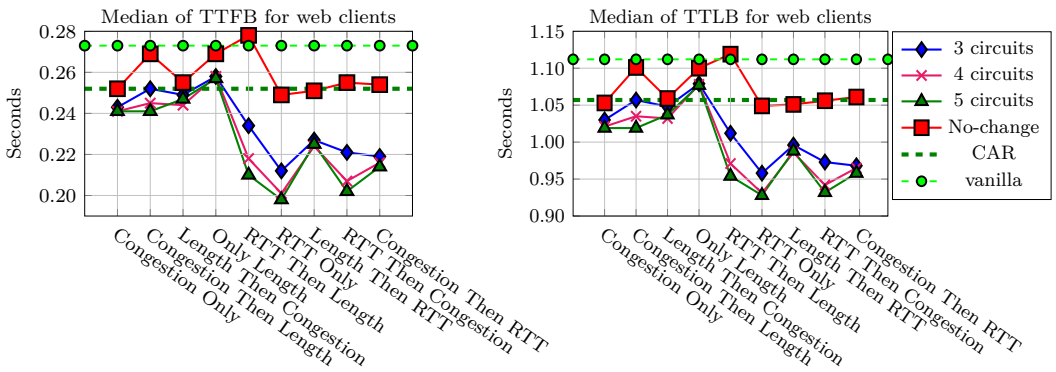
Since these circuit selection mechanisms are deterministic, given a set of candidate circuits, only one circuit from a set will be be used. These strategies will exploit the best circuit for the full 10-minute window that the circuit can be used. Since this means that the other circuits will go unused, we have the OP close any circuits that go unused for five minutes after their creation, leading to new circuits being opened. By itself, this might improve performance, as inferior circuits are closed in favor of untested circuits that may be better (or worse).

3.5 Circuit Selection Performance

We now evaluate the nine methods of selecting circuits for stream attachment and compare them with Tor and CAR.

3.5.1 Network Configuration

We largely follow the experimental procedures suggested by Jansen et al. [32] and describe them here in brief. Shadow runs actual Tor code for accurate modeling; we used Tor version 0.2.5.12, modifying it as necessary to implement our methods and CAR. To generate a realistic Tor network topology, Shadow comes with topology generation tools that model a private Tor network based on a validated research study [32]. We used these



(a) TTFB (b) TTLB

Figure 3.2. **Circuit Selection:** TTLB and TTFB for web clients..

tools and data from the Tor metrics portal to generate our private Tor network. Our Tor network includes 1100 clients, 220 Tor relays (52 exit relays, including exit-guard relays, and 49 guard relays), three directory authorities, and 220 HTTP destination servers.

Shadow uses an underlying topology that models the Internet. The default topology shipped with Shadow is very small, consisting of only 183 vertices and 17,000 edges, and is not a good representation of the Internet. For all the simulations in this paper we used the same Internet topology that was used by Jansen at al. [45]. This topology is provided by techniques from recent research in modeling Tor typologies [32, 46], traceroute data from CAIDA [47], data from the Tor Metrics Portal [48] and Alexa [49], and it includes 699,029 vertices and 1,338,590 edges. In our simulation, we tried different ratios of clients to relays, i.e. different congestion levels, and different average packet loss rates in the Internet topology and compared our results with Torperf data [48]. We found that a clients-to-relays ratio of 5:1 with 0.0025% packet loss provides us comparable results on TTFB and TTLB with Torperf data.

Our clients run Tor code in client-only mode and are distributed around the world in line with Tor usage statistics. We have two types of clients in our experiments: web clients and bulk clients. The 900 web clients download 320 KiB of data (the average page

size [50]) and simulate web-surfing behavior by waiting between 1 to 20 seconds uniformly at random before starting the next download. The 100 bulk clients download 5 MiB of data without pausing between the end of a download and starting the next one.

3.5.2 CAR: Congestion-Aware Routing

To compare our methods, we also simulated CAR, the circuit selection technique of Wang et al. [28]. They proposed opportunistic and active probing techniques to measure RTTs, which allows them to compute congestion times according to Equation 3.2, and they use these measurements to mitigate congestion using both an instant response for temporary congestion and a long-term response for low-bandwidth conditions. In our simulation, we follow the method of Wacek et al. [31], who also simulated CAR and ignored the long-term response due to its small impact on performance.

When attaching streams to circuits in CAR, we randomly select three circuits from the circuit list and pick the one that has the smallest mean congestion time from the five most recent measurements. If the mean of last five congestion times is more than 0.5 seconds for a circuit, we stop using the circuit for new streams.

3.5.3 Performance Results

Figure 3.2 shows the median time-to-first-byte (TTFB) and time-to-last byte (TTLB) for web clients. *Vanilla* represents unmodified Tor circuit selection, and *No change* represents the case where we do not modify the number of circuits from Tor, which typically has one or two circuits available at a time.

As shown in Figure 3.2, RTT is the best criterion to choose the circuit, with *RTT Only* as the best method overall. *RTT Only* has 15% lower TTFB than CAR (22% lower than *Vanilla*) for three circuits and 22% lower TTFB than CAR (27% lower than *Vanilla*) for five circuits. *RTT Only* also has 9% lower TTLB than CAR (13.8% lower than *Vanilla*)

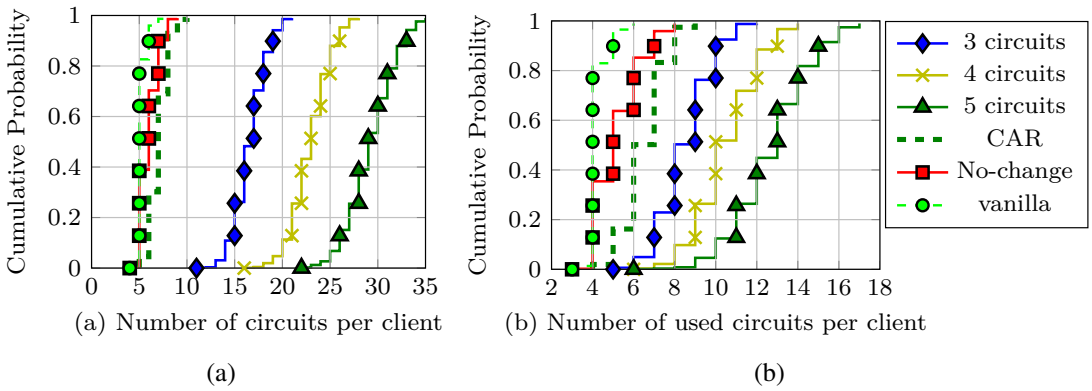


Figure 3.3. CDF of the number of circuits created (a) and used (b) for web clients..

for three circuits and 12% lower TTLB than CAR (16% lower than *Vanilla* for five circuits. We speculate that RTT is the best criteria because it effectively captures both propagation delays and congestion time (queuing delays and transmission delays).

As expected, CAR is better than *Vanilla*, and *Congestion Only* with *no change* in the number of circuits performs the same as CAR, as both use the same criteria. Length turns out to be less effective compared to RTT or congestion times, particularly when the number of circuits is small. We note that *Length then RTT* performs fairly well for three or more circuits. Length may be suitable for gauging broad performance information, such as comparing a circuit with multiple intercontinental hops to one with no such hops, but poor at predicting the best circuit otherwise.

The TTFB and TTLB for *RTT Then Congestion* are slightly better than *Congestion Then RTT*, which indicates that RTT can narrow down candidate circuits better than congestion times. Results of both *RTT Then Congestion* and Congestion Then RTT are worse than *RTT Only*, which shows that mixing congestion time with RTTs will not provide better performance than using RTT by itself.

3.5.4 Circuit Creation Analysis

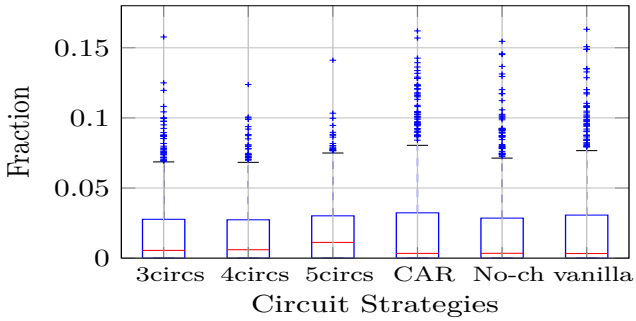
In our circuit selection strategies, we build more circuits than Tor's normal behavior, so it is important to understand the load this imposes on the network. Unfortunately, Shadow does not provide results regarding the load on nodes. To estimate changes in load, we compare the strategies based on the number of created circuits and used circuits. To see how many circuits our clients build, we simulate the circuit selection strategies in Shadow for one hour of simulated time, which leads to about 40 minutes of activity after 20 minutes of initialization. We extract the number of general-purpose circuits built by our web clients. Note that this does not including circuits built for hidden services or downloading the consensus, which is a consistent load across all schemes.

Figure 3.3 shows the CDF of created *general purpose* circuits and the CDF of used circuits, the circuits actually being used for transferring the data. We show results for web clients in *Vanilla*, CAR, and *RTT Only* with the same number of circuits as Tor, as well as *RTT Only* with $N = 3, 4, 5$ circuits. The median number of created circuits in *RTT Only* is 17, 23, and 29 circuits as we increase the number of circuits from three to five. *RTT Only* leads to building so many circuits due to proactively checking every second that there are N circuits available for each recently used port, plus killing unused circuits after five minutes. Fig. 3.3.b shows how many of these created circuits have been used in transferring data. The median for used circuits in *vanilla*, CAR, and *RTT Only* with no change in the number of circuits is around four circuits, which means that they use all the circuits created and attach some stream to them. The median in *RTT Only* is 8, 10, and 13 circuits, respectively.

3.6 Security Analysis

In this section we examine the security of these circuit selection strategies, considering both relay-level and network-level adversaries. Our performance results show that

RTT Only outperforms all the other circuit selection strategies and CAR. Therefore, in this section, we focus on the security analysis of *RTT Only*.



(a)

Figure 3.4. *Relay-level adversary*: Distribution of compromise rates..

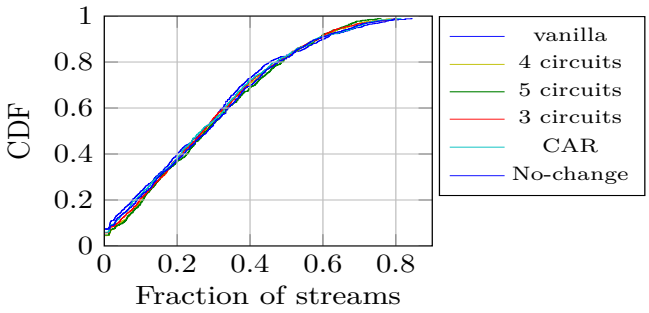
3.6.1 Relay-Level Adversary

In the relay-level adversary model, we assume that the adversary runs both guard and exit relays in the hope that his relays simultaneously occupy the guard and exit positions in some circuits. If the adversary can sit on the exit and guard position on a circuit, he can apply a traffic correlation attack and link the client to her destinations. These circuits and the streams attached to them are called compromised circuits and compromised streams, respectively. To analyze the security of the *RTT Only* strategy, we need to have access to RTTs (which include propagation delays, queuing delays, and transmission delays), which means we need to simulate a whole network. For this purpose, we again use Shadow to simulate the Tor network, and we use the same Tor network configuration as our performance evaluations in Section 3.5.1, which consists of 52 exit relays, including exit-guard relays, and 49 guard relays.

For the relay-level adversary, we randomly mark 10% of our guard bandwidth and 10% of our exit bandwidth as malicious guards and malicious exit relays in the network.

Then we simulate CAR, *Vanilla*, and *RTT Only* with an unchanged number of circuits and then *RTT Only* using three to five circuits. We run 10 simulations, where the malicious guards and exits change in each run. 10 simulations for each case is reasonable considering that we have 52 exit relays, 49 guards, and simulations taking 11 hours.

Figure 3.4 shows box plots for the stream compromise rates for clients. The median of compromise streams is almost the same for *vanilla*, CAR, and *RTT Only* with *no change* in the number of circuits because the clients build almost the same number of circuits. As the number of circuits increases, the median of compromised streams rate increases due to the increase in circuits created by the clients. When the number of circuits increases, the chance of creating a circuit that has malicious relays on its guard and exit positions increases. As we see, *RTT Only* with five circuits has the highest compromised streams with a median of 1% while *vanilla* and CAR have a median compromised streams of 0.3%.



(a)

Figure 3.5. *Network-level adversary*: CDF of compromise rates..

3.6.2 Network Level Adversary

In the network-level adversary model, we assume that the adversary controls an Autonomous System (AS). If the entry traffic (traffic between the client and guard) and exit

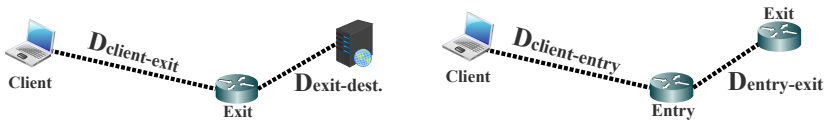
traffic (traffic between the exit and server) traverse over a common AS, that AS can apply traffic correlation attack and link the client to her destinations.

To analyze the security of circuit selection strategies, we used our Shadow simulation results from the relay-level adversary for CAR, *Vanilla*, and *RTT Only* with an unchanged number of circuits and then *RTT Only* using three to five circuits. Because we did not add any relays to the network in evaluating the relay-level adversary model, as we only marked existing relays as malicious, we can re-use these results for this analysis. For each circuit selection approach, we extract all the generated streams by clients. The simulations generated approximately 730,000 streams for each circuit selection approach, or about 700 streams per client. For each stream, we used the algorithm proposed by Qiu and Gao [43] to infer the AS paths between clients and guards and between exits and servers. This algorithm exploits known paths from BGP tables to improve the accuracy of the inferred paths. In measuring the compromise rates, we consider the possibility of an asymmetric traffic correlation attack that can happen between the *data* path and *ack* path, which is one of the RAPTOR attacks proposed by Sun et al. [44].

Figure 3.5 shows the cumulative distribution of compromise rates for each strategy. As we see, when we increase the number of circuits from three to five, the the median compromise rate increases from 27.2% to 28.1% while the compromise rates of CAR and *vanilla* are 26% and 27%, respectively. CAR performs 1% better than *Vanilla*. These results show that using RTT and increasing the number of circuits even up to five circuits have a modest effect on the security against a network-level adversary.

3.7 Relay Selection

In this section, we describe a method for path selection in which we assign weights to relays based on a combination of bandwidth and geographical distances. This approach



(a) Exit (b) Entry

Figure 3.6. Distance for relays for each position..

extends the idea of Tor path selection, which uses weights based on various factors in probabilistic relay selection. The goal of the combined weighting approach is to build circuits that still have high bandwidth relays, ensuring load balancing and good throughput, but also relatively shorter paths between the client and her destinations.

3.7.1 Weight Function

In a large and growing network like Tor, which consists of around 7000 nodes as of August 2016, examining all possible paths to find ones with these characteristics would be expensive. Clustering relays into geographic areas, as in LASTor [30], can lead to uneven distribution of bandwidth between clusters. Instead, we approach the problem much like Tor's current algorithm by selecting one relay at a time, starting with the exit node, then the entry node, and finally the middle node. This greedy approach may miss the optimal path, for some definition of optimal, but our design aims to select from a wide range of paths with good performance and to avoid poorly performing paths. A broader selection of paths should help us maintain anonymity.

We calculate the weight of each relay using following function:

$$w = \alpha \times w_B + (1 - \alpha) \times w_D$$

where w_B is a measure of the relay's bandwidth and w_D is a measure of distance. In this function, α is parameter that we can use to tune the share of bandwidth and distance in the weights. As α increases, the importance of bandwidth to the weight increases, and as α

decreases, the importance of distance to the weight increases. w_B and w_D for relay *i* are computed as follows:

$$w_{Bi} = \frac{B_i}{B_{max}}, \quad w_{Di} = 1 - \frac{D_i}{D_{max}}$$

Here, B_i is the relay's weighted bandwidth, and B_{max} is the maximum weighted bandwidth among all relays. Tor assigns weights for each position in the circuit, and these weights bias the relay selection for circuits to distribute more load to higher-bandwidth relays. D_i is a distance that is computed differently depending on the selected relay's role in the circuit as exit, middle, or entry. The maximum value of D_i over all relays is D_{max}. We subtract the ratio from 1 so as to weight short distances more than long ones. Note that both w_B and w_D will be between 0 and 1.

Figure 3.6 shows how we compute the distance for relays for exit and entry position in the circuit. We seek to minimize total distance from the client to the destination by minimizing the intermediate pairs of distances that are added by each relay in the sequence used by Tor: exit, entry, middle. Intuitively, selecting one of these relays with a large distance to its neighbors extends the path away from a straight line between client and destination, which would theoretically be the ideal path.

Since we use geographic locations, we compute D using the great-circle distance between two points on a sphere from their longitudes and latitudes. In choosing the circuit's exit node, we compute D for all relays as follows:

$$D_{exit} = (1 - \lambda) \times D_{client-exit} + \lambda \times D_{exit-dest}$$

For the circuit's entry node:

$$D_{entry} = \lambda \times D_{client-entry} + (1 - \lambda) \times D_{entry-exit}$$

And for the circuit's middle node

$$D_{middle} = D_{entry-middle} + D_{middle-exit}$$

(a) Large value of λ (b) Small value of λ

Figure 3.7. The effect of λ..

D_{max} in w_D is the maximum computed D among the set of relays for each position (exit, entry, or middle). The use of D_{max} and B_{max} ensure that w_D and w_B both range between 0 and 1 for more straightforward calculations.

λ is a tuning parameter that enables us to change the share of the distance between different nodes on the path. As shown in Figure 3.7, as λ goes up, the entry nodes and exit nodes move toward the source and destination, respectively, and a large portion of the path between the source and destination is covered by inter-relay connections. In this case, the selected guards are close to the clients which can decrease the threat of network level adversaries. In particular, it causes the AS paths between clients and guards to be shorter and involve fewer ASes, thereby decreasing the chance that a common AS appears on both sides of the traffic, i.e. between the client and guard and between the exit and server. As λ decreases, the path's inter-relay portion shrinks.

To evaluate how close the short paths selected by our algorithm are to optimal, we used a scaled-down Tor network, with 147 exit nodes, 700 middle nodes, and 170 entry nodes, a user located in the central US, and 100 destinations from the Alexa Top 100 websites [49]. We found short paths between the user and all destinations with our method for different values of λ. To measure the average distance between our method from the actual shortest path to each of these 100 destinations (d), we used the mean absolute percentage deviation (MAPD):

$$dev_\lambda = \frac{1}{100} \sum_{d=1}^{d=100} \frac{|L_{\lambda d} - L_d|}{L_d}$$

where $L_{\lambda d}$ is the length of the shortest path found by our method for a given value of λ, and L_d is the length of the shortest path to destination d. Figure 3.8 shows the average deviations dev_λ for different values of λ, and we see that $\lambda = 0.5$ has the lowest deviations at just 0.19% longer on average. This indicates that our greedy algorithm produces short paths close to the optimal ones.

3.7.2 Preemptively built circuits

When computing distances, we need to know the location of the final destination. The destination addresses can be either an IP address or a DNS hostname. If the address is IP, we can find the destination's location by using IP geolocation databases (we use Maxmind [51] in this paper) and find the shortest path to these destinations. But most of time, the addresses are DNS hostnames. To find the location, we first need to perform DNS resolution to get the IP address. In the DNS resolution process, the party performing DNS lookup returns the closest content provider or replica sever to itself, which in Tor's case means the closest ones to the exit nodes.

In Tor, however, the client saves time by having a number of circuits already built and available for new connections. Building a new circuit takes time for numerous protocol messages and public-key cryptographic operations. Wacek et al. showed that LASTor, which builds new circuits once the destination location is discovered, suffers from significant added delay due to these delays [31].

To solve this issue, we build circuits in advance that shorten the path between the source and the most popular destinations online. To identify popular destinations, we use the Alexa Top 1000 websites [49]. Because some of these sites use CDNs or replica servers, we visited them from different places in thew world. In particular, we selected 13 planet-lab nodes based on Tor users' statistics [52]. From each node, we visited all of the 1000 sites, got the address of all fetchable elements in their front pages, and resolved their addresses

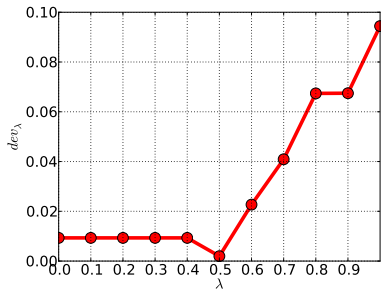


Figure 3.8. Average added distance over the shortest path for different values of λ..

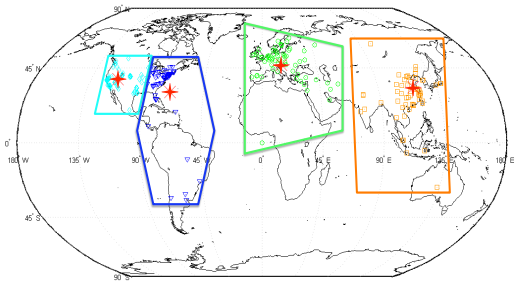


Figure 3.9. The location of the existing servers in Alexa Top 1000 websites. Red stars show the cluster centroids..

if needed. For example, from our planet-lab node on the US west coast, we got 46,306 hostname addresses, leading to 2894 unique IP addresses from 346 unique locations. We clustered the obtained locations into four clusters using the k-means algorithm (we got better performance for four rather than with three or five clusters), and we use the centroids of these clusters as a target destination. Figure 3.9 shows all obtained locations and our cluster centroids. From these four target destinations, we mark the one closest to the client as the default destination, i.e. when we have no other information, we assume that the users is more likely to visit sites located closer to her. We have the client build circuits in advance of their use with short paths to these four target destinations. We check our circuit list once per second to ensure that we have at least one circuit to each of these destinations and ensure that a new connection can be handled as quickly as possible.

3.7.3 Guard Selection

In Tor, each client selects and uses one guard consistently for a period of nine to ten months. This means that for selecting the path, the guard relay is already selected, before the exit. We thus cannot use the exit's location to help pick the entry and must modify our algorithm. In selecting the guard, instead of using the exit node's location, we use

the closest of the four target destinations to the client to compute D_{entry}. D_{entry} will be computed as:

$$D_{entry} = \lambda \times D_{client-entry} + (1 - \lambda) \times D_{entry-target}$$

This can reveal some information about the client's location, e.g. through fingerprinting attacks that identify the guard from the exit [11]. Since we only have four popular destinations, however, the anonymity sets for clients' location will be quite large. We evaluate the security of our design in Section 3.9. Further performance improvement can be achieved by using $D_{guard-exit}$ instead of D_{client_exit} in computing D_exit because we have already selected the guard relay and know its location. This helps us to not have long circuits in case the guard is relatively far from the client.

During this research, we found a bug in the Tor source code that was causing Tor clients to choose guards from all the available relays, not the ones with the guard flag. This harmed both anonymity and performance. The bug was reported to the Tor project[4] and was fixed in Tor version 2.7.6. We use the corrected code in all of our simulations.

3.7.4 Attaching streams to the circuits

Our relay selection mechanism tends to build short circuits to the popular destinations. As a stream request a circuit, Tor client should pick a circuit among all preemptively built circuits or build a new one if there is no circuit available. Tor does not consider any performance metric in picking the best circuit among the preemptively built circuits. In order to find the fast circuits among preemptively built circuits, we first select the two circuits with lowest RTTs and then pick the shorter one. We use the opportunistic circuit measurements and latency model proposed by Wang et al. [28] to measure the circuit RTT. This circuit selection mechanism helps us to choose circuits which are fast and short.

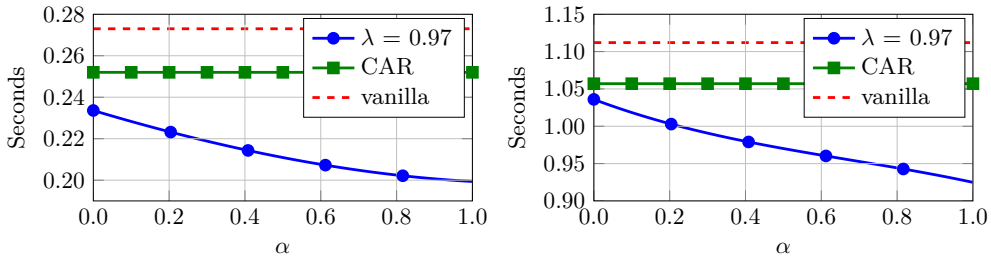
[4] `https://trac.torproject.org/projects/tor/ticket/17772`

3.8 Performance Evaluation

In this section, we evaluate the performance of our proposed relay selection method compared with Tor and congestion-aware routing (CAR), the current state of the art [31]. We evaluate the performance of our method as α and λ vary.

The effect of α. For evaluating different values of α, we set $\lambda = 0.97$ and vary α from 0.0 to 1.0, we have chosen λ high to stretch circuits between the clients and destinations. Figure 3.10 shows the median of web clients' TTLB and TTFB with respect to α, where we plot only the median of the results for readability. Changing α from 0.0 to 1.0 yields 7% to 24% improvement in web clients' TTFB compared to CAR, and 2% to 12% improvement in web clients' TTLB compared to CAR. Relay bandwidth is more important in improving performance, which matches findings from Wacek et al. [31]. We will explore the trade-offs in security in Section 3.9.

The effect of λ. Parameter λ controls the elasticity of the path. As λ increases, the circuit stretches out, with guards moving toward the clients and exits moving toward the destinations. To evaluate the effect of λ, we set $\alpha = 0.0$. We find that performance changes less than 2% as λ varies, and it is best for λ of 0.4-0.5.



(a) TTFB (b) TTLB

Figure 3.10. Median TTFB and TTLB for web clients.

Strategy	Gini Coef.	Entropy
vanilla	0.725	9.702
$\alpha = 1.0$	0.724	9.713
$\alpha = 0.9$	0.590	10.326
$\alpha = 0.8$	0.490	10.609
$\alpha = 0.7$	0.444	10.735
$\alpha = 0.6$	0.401	10.802
$\alpha = 0.5$	0.370	10.872
$\alpha = 0.15$	0.335	10.933
$\alpha = 0.1$	0.338	10.944
$\alpha = 0.0$	0.341	10.945

Table 3.1. **Relay adversary.** System-wide security results.

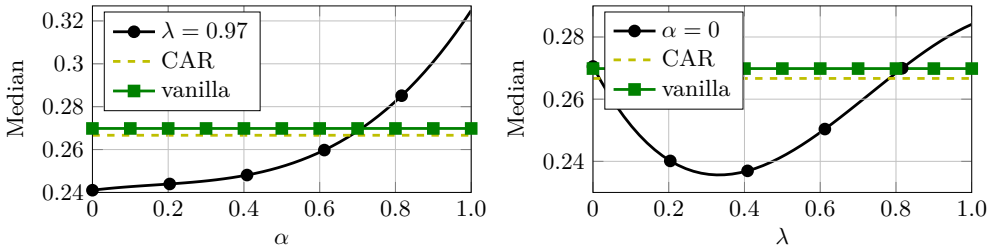
3.9 Security analysis

We now examine the impact of our path selection strategy on anonymity using three models. First, we study broad system-wide measures of anonymity. We then examine path compromise rates in the presence of AS adversary. Finally, we study a set of targeted attacks in the relay level adversary.

3.9.1 System-wide Security Metrics

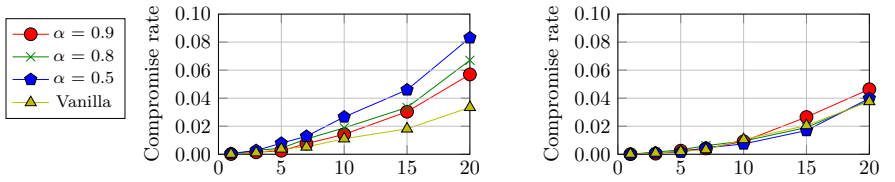
We simulated the proposed path-selection strategies in a 2127-relay model of the Tor network, built by sampling approximately one-third of the nodes of each type (exit, entry, middle) from a descriptor file from December 2015. In our simulations, 200 clients are placed according to statistics about users from the Tor metrics portal, and each client constructs 27,000 paths. In our location-based approaches, the clients select paths using the four target destinations as described in Section 3.7.2. We created in total 5.4 million paths for different values of α, with $\lambda = 0.97$.

To measure anonymity, we focus on end-to-end traffic confirmation attacks in which the adversary controls both the exit and entry relays in a circuit. We measured the Gini co-

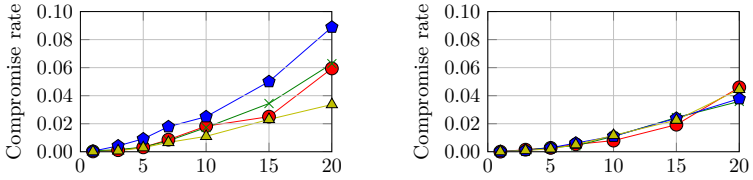


(a) As α varies (b) As λ varies

Figure 3.11. **AS Adversary.** The median stream compromised rate as α and λ vary.



(a) Targeted Clients (b) Targeted Destination



(c) Targeted Dest. and Client (d) Non-Targeted

Figure 3.12. **Targeted Attacks:** Fraction of circuits compromised. X-axis shows the percentage of exit and guard bandwidth controlled by the attacker..

efficient and Shannon entropy of the exit-entry combinations occurring on selected circuits. The Gini coefficient is a measure of the equality of relay selection, where 0 represents pure equality (i.e. each relay is selected uniformly at random) and 1 represents a state of complete inequality (i.e. a given relay is always selected) [27, 31]. Table 3.9.1 shows the results for Gini coefficient and entropy.

In our static path selection method, we cannot produce equivalent results for CAR, since circuits dynamically change in their method. Wacek et al. report that CAR has a lower (i.e., better) Gini coefficient than Tor but slightly lower (i.e., worse) entropy [31].

As shown in Table 3.9.1, as α decreases, the Gini coefficient decreases and the entropy increases. As we expect, for α = 1.0, which selects the relays only based on bandwidth, the results are nearly the same as *vanilla*. The most dramatic difference in anonymity occurs for higher values of α, e.g. from $\alpha = 1.0$ and $\alpha = 0.8$, where the Gini coefficient drops from 0.724 to 0.490 and entropy rises 0.9 bits.

Since α means increasing the share of distance in the selection weights, we see that emphasizing distance in the weights improves the system-wide security metrics. On the other hand, according to Figure 3.10, small values of α offer lower performance. We thus face a trade-off between security and performance, where decreasing α improves security but offers less performance benefits. Values of α between 0.8 and 0.5 offer both good security and performance, with a Gini coefficient of between 0.370-0.490 and almost 20% improvement in TTFB compared to CAR.

3.9.2 AS Adversary

In this section, we evaluate the security of our approach in the presence of AS-level adversaries. We use Shadow with the same configuration as previous section and carry out simulations for different α and λ values.

For evaluating the effect of α, we fix $\lambda = 0.97$ and vary α from 0.0 to 1.0. For each value of α, we extract all the generated streams along with their attached circuits in the simulation. For all the streams, we find the AS paths between the clients and guards (guards and clients) and between the exits and the destinations (destinations and exits) using the algorithm proposed by Qiu and Gao [43]. We consider the possibility of an asymmetric traffic correlation attack that can happen between the *data* path and *ack* path. Figure 3.11(a) shows the median stream compromise rates. As we see, by increasing α from 0, the compromise rate starts increasing until α reaches around 0.7, where we have

the compromise rate equal to *vanilla*. The compromise rate in CAR is slightly better than *vanilla*.

To evaluate the effect of λ on the security, we set $\alpha = 0.0$ and vary λ from 0 to 1.0. Figure 3.11(b) shows the median compromise rate as λ varies. When λ increases, the compromise rate decreases till it reaches 0.4 after that it keeps increasing.

3.9.3 Targeted Attacks in Relay-Level Adversary

To further explore how path selection strategies perform against attacks in the relay level model, we now examine four types of attacks in the network and measured how often adversaries can compromise a circuit. We assume that a circuit is compromised if both the exit and entry nodes are controlled by the adversary.

In the targeted attacks, we consider a high-bandwidth adversary that owns a few high-bandwidth relays such that its bandwidth is a considerable fraction of the network's total bandwidth. In particular, we start with our 2127-relay model of the Tor network as described in Section 3.9.1. We select a random bandwidth in the range [20 MiB/s, 220 MiB/s], where 220 MiB/s is the maximum bandwidth in our model network. A malicious relay with this bandwidth is added to the Tor network, where the location of the malicious relay is based on our attack strategies. This process is repeated until the target attacker bandwidth for this *run* of the experiment is reached. For each of 200 runs at each bandwidth setting, we place one client into the network using a location based on Tor metrics data and have it pick 9,000 paths for each of our tested strategies. We use the four popular destinations as described in Section 3.7.2 for our strategies. In our evaluations, we consider four different attack strategies as follows:

1. *Targeted Clients*: In this attack strategy, in each run that we add the client and malicious relays, all the malicious guards are located in the exact location of the client, just as if

the adversary could run all of his guards in the client's room. The malicious exit relays are randomly placed in locations based on the geographical distribution of Tor relays.

2. *Targeted Destination*: In this attack strategy, in each run that we add the malicious relays, all the malicious exits are located in the exact location of the one of the randomly selected popular destinations, just as if the adversary could run all of its exit relays in the same server room. The malicious guard relays are randomly placed in locations based on Tor relays geographical distribution.
3. *Targeted Destination and Client*: In this attack strategy, in each run that we add the client and the malicious relays, all the malicious exits are located in the exact location of the one of the randomly selected popular destinations, and all the malicious guards are located in the exact location of the client.
4. *Non-targeted*: In this attack strategy, in each run that we add the malicious relays, all the malicious relays are randomly placed in locations based on the geographical distribution of Tor relays.

Figure 4.9 shows the fraction of compromised paths with respect to the percentage of total bandwidth controlled by the adversary's relays for *vanilla*,α = 0.5,α = 0.8, and α = 0.9.As shown in Figure 4.9, the compromise rate for *Targeted Destination* is almost the same as compromise rates in *vanilla*. For *Targeted Client* and for *Targeted Destination and Client*, the compromise rates for α = 0.5,0.8, and 0.9 are worse than *vanilla*, and as the adversary's bandwidth fraction increases, the gap between them and *vanilla* increases. For both *Targeted Client* and for *Targeted Destination and Client*, α = 0.8 has almost the same compromise rate as α = 0.9 but better compromise rate than α = 0.5. For *Non-targeted* we observed the same compromise rate as *vanilla* for α = 0.5,0.8, and 0.9 because in this attack malicious relays are randomly located in the network and all relay selection methods pick the malicious relays with the same probability.

We also note that this is a trade-off with the modest, but wide-spread, security benefits of using $\alpha = 0.8$ on Gini coefficient, entropy, and compromise rates compared with $\alpha = 1.0$, for example. Greater emphasis on bandwidth leads to better performance and more resilience to targeted attacks, while greater emphasis on distance leads to more diffuse spreading of load on the network.

3.10 Discussion

In this section, we discuss the implications of our findings and the scope for future research.

Circuit Selection. We evaluated the impact of different number of pre-built circuits on Tor performance, and found that having at least three pre-built circuits ready results in a significant improvement compared to *vanilla* Tor. Preparing more than three circuits, however does not provide much additional benefit and may also add more load on the network. Our circuit selection mechanisms also kill unused circuits after five minutes, which raises the rate of exploring for better circuits.

Relay Selection. In relay selection, combined weighting seems to provide a trade-off of performance and anonymity. As the weights emphasize on the bandwidth, α is high, combined weighting provides higher performance. On the other hand, higher values of α could not provide diverse paths. As α goes down and the weights are inclined toward the distances, the performance improvement decreases, but the created circuits are more diverse. Low values α suffer from a greater chance of targeted relay-level attacks. Overall, we think that combined weighting with $\alpha = 0.8$ seems to provide the best trade-off of performance and anonymity. The best value of α may vary with network configuration, bandwidth distribution, geographical dispersion of relays, and the client's location. We will examine setting α more carefully in future work.

Nearby guards. Our evaluations showed that the proposed defense, picking guards close to the clients, does not effect an AS-level adversary. The AS path between the clients and guards is not highly correlated with the geographical distance between them. The AS path length between the guards and clients depend on the clients' networks, guards' networks, and their ASes relationships with other ASes. Moreover, the clients and guards are not uniformly distributed on the globe and on the network. For example, a single AS, AS16276, is contributing more than 170 guards to the Tor network, which is 16% of all the guards in January 2015. The other issue is guard rotation, as currently Tor clients change their guard after 9 to 10 months. In our 10-month TorPS simulations, the median number of guard changes for clients was five times, with a minimum of two times and a maximum of 29 times. Thus, even if the client is secure due to the short AS path, after guard rotation, she may pick a guard that has a long AS path length and get compromised.

CHAPTER 4

Defense against Guard Fingerprinting attacks

4.1 Introduction

Tor allows clients to create anonymous connections to their desired destinations via three-hop encrypted channels called *circuits*. A circuit is built over a path of three relays, an *entry*, a *middle*, and an *exit*, selected from among the thousands of volunteer relays distributed across the globe. In Tor, no single relay in the circuit nor any third party in the network should be able to link the source with the destination.

Since relays are run by volunteers, however, it remains a risk that multiple relays on a circuit are run by a single entity who could then break the user's anonymity. In fact, if all relays on the circuit were picked at random every time, a Tor user would be rolling the dice with her privacy every few minutes. Most circuits would be fine, but eventually she would roll a pair of malicious relays and lose her anonymity. To prevent the majority of users from getting compromised, Tor fixes the client's entry node to be the same in every circuit for up to nine months. If this entry node, called a *guard*, is honest and does not get compromised, then the client's identity cannot be directly discovered by malicious relays while the guard is still being used [53].

A key design decision around the use of guards is how to assign guards to users. If a user picks a guard with very low bandwidth, as an extreme example, then not only will her performance be poor over an extended period of time, she may be the only user regularly using that guard and can thus be profiled [9, 39, 11, 54]. This is known as *guard fingerprinting*. More generally, there are several anonymity and performance considerations for picking guards that have only recently been explored [55, 56, 57, 58].

One solution to the guard fingerprinting problem is to group all guards into *guard sets* [57] and have each client pick one of the guard sets and use the guards in this guard set for the first hop on all of its circuits. Hayes and Danezis [14] proposed the first guard set algorithm for use in Tor. This algorithm uses guard relays' bandwidth as the key criterion in forming guard sets, such that all sets have almost the same amount of bandwidth. They also presented techniques for maintaining the guard sets when there is churn.

Contributions. In this paper, we first demonstrate that the algorithms proposed by Hayes and Danezis have vulnerabilities that allow an attacker to compromise many guard sets in the presence of churn over time (§4.3). In particular, we describe attacks that leverage the fact that the attacker controls the amount of bandwidth it makes available for a given guard node. With these attacks, a low-bandwidth adversary controlling 1% of total guard bandwidth can infiltrate around 40% of all guard sets within four months, and a high-bandwidth adversary controlling around 25% of total guard bandwidth can infiltrate 90% of guard sets.

To address these issues, we propose a new guard set design (§4.4) that uses location in the Internet topology as the basis for building a hierarchy on top of the sets. Using this hierarchy, sets are built and maintained using guards that are topologically close to each other in the Internet. This limits an attacker's ability to compromise guard sets beyond whatever Internet locations he has access to. While bandwidth can be easily manipulated, many potential attackers will have a limit on the possible Internet locations of their guards.

We evaluate the security of this approach against attackers who control a fraction of the guards, with varying resource levels, using one network in the Internet (§4.5.1.1). Against a single malicious guard, the compromise rate after one year of running the attack is 0.044% compared to 0.076% for the prior approach. Against an attacker who controls 10% of Tor's guard bandwidth, the compromise rate compared to the prior approach dropped

from 53% to 10% after one year and less than half of the rate for the current Tor design (23%). Against a botnet adversary, which inherently has a presence in more AS locations, the compromise rate fell from 53% to 37% compared to the prior approach after one year. Moreover, the fraction of compromised targets in our approach dropped from 98% to 44% compared to the prior approach in a targeted attack scenario (§4.5.1.2).

We also evaluate our approach against attackers who control one *Autonomous System (AS)* in the Internet (§4.5.1.3). We find that our approach has very similar results to both Tor and the prior work. Additionally, we merge our guard set design with DeNASA [59], a recently proposed AS-aware path selection algorithm, and show that the rate of streams being vulnerable to attack drops 80%.

Beyond this, we evaluate a number of other aspects of the proposed design (§4.5.2), including the sizes of anonymity sets, the bandwidth distribution among guard sets, and network performance. We conclude with a discussion (§4.6) of deployment and other issues to be addressed in future work.

4.2 Background

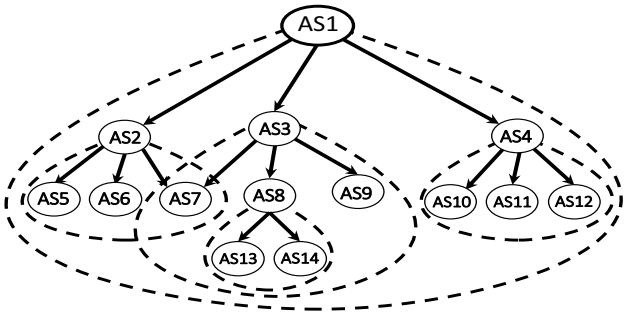
In this section, we briefly overview the AS structure of the Internet and the Tor anonymity system, and we then discuss related work.

chapter4

4.2.1 Autonomous Systems

Our approach makes use of the structure of the Internet topology, so we describe the necessary concepts here.

The network layer of the Internet is composed of *Autonomous Systems (ASes)* that are linked together by high bandwidth lines and fast routers. Each AS is owned and operated by one authority, such as a government, university, or Internet service provider. ASes



(a)

Figure 4.1. **Customer cones**: The dashed lines show customer cones, the solid arrows are provider-to-customer links..

contain a set of servers that are linked together in a LAN and are assigned IPs from an IP prefix that is unique to that particular AS. Relationships among ASes have been formed based on a variety of economic and political constraints. Using publicly available BGP table data, Gao [60] introduced a method that abstracts these relationships in three types: customer-to-provider (*c2p*), provider-to-customer (*p2c*), and peer-to-peer (*p2p*). In a c2p or p2c relationship, the customer provides monetary payment to the provider in exchange for the provider providing bandwidth to the customer. In a p2p relationship, the two ASes save monetary resources by exchanging traffic between one another on a quid-pro-quo basis.

Viewing ASes and their relationships in graph theoretic terms, we have a forest of trees, with backbone providers at the root nodes and customers as leaves. We can then define a *customer cone*, the set of ASes that can be reached from the root AS by only following the p2c links. For example, AS A's customer cone consists of AS *A* plus AS *A*'s customers, plus AS *A*'s customers' customers and so on [61]. Figure 4.1 shows an example of how customer cones work, where AS1 has the biggest cone in the example and contains all the ASes shown, while AS8 has only AS13 and AS14 in its customer cone. The number of ASes in a customer cone and the number of unique IP prefixes advertised by these ASes

are good metrics for ranking the size and importance of an AS [62]. Our guard set design groups the guard ASes that are in the same customer cone in the same set.

The underlying customer cone in our design can be built by different methods. The main requirement for our system are that the cones should be relatively stable over time compared with bandwidth fluctuations in the guard sets. We chose to use the *recursive customers* method [62, 61]. In this method, the customer cone of each AS is built by recursively visiting ASes reachable from that by p2c links. The customer cone of an AS is a subtree of customer ASes that can be reached from that AS. The size of a customer cone is the number of customer ASes in that cone. For the the customer cones in our experiments, we use the AS relationships provided by CAIDA for July 2015 [63].

4.2.2 Tor Overview

Tor is a volunteer-operated network that provides anonymity and privacy online. Tor has about 2 million daily users and 7,000 relays. Relays in Tor network, called *Onion Routers* (ORs), are run by volunteers who donate their bandwidth. ORs provide information about their donated bandwidth, IP address and ports, and *exit policies*—the addresses and ports they are willing to be connected to external Internet destinations—to a small group of servers called *directory authorities*. The directory authorities assign flags to some of the ORs based on their availability, bandwidth and exit policies and then they mutually agree upon a list called the *consensus* of all the information about all the ORs.

A Tor client, called the *Onion Proxy* (OP), first contacts one of directory authorities or their mirrors and downloads the consensus to get the current status of the Tor network. Since the Tor network is dynamic, with relays regularly joining and leaving, the directory authorities update the consensus hourly. The OP uses the consensus information to select a path of three relays to use in communicating with its destinations.

Once the OP picks this path of ORs, it then sets out to build a *circuit* of layered cryptographic connections through this path. Since relays in the circuits are selected from all the volunteer relays, it is possible that an adversary, who runs some guard or exit relays in the network, sits in the exit and entry position of some path. Such an adversary can observe the entry and exit traffic, correlate them, and link the client to her destinations [39, 11, 46, 44]. If the client chooses new relays for each circuit, she will eventually build a circuit in which the adversary's relays are in its entry and exit positions. To reduce the occurrence of this attack, Tor clients stick to using a single *guard* relay for the first relay on every path for months [57]. Guards should thus be stable, so that the client can rely on the guard to be available whenever it connects to Tor, and reasonably high bandwidth to prevent the guard from becoming a significant bottleneck to performance.

Directory servers keep track of relays' bandwidth and availability in the network and assign Guard flags to the relays that have the following criteria [56]:

- Have been continuously running for longer than 12.5% of the relays or at least eight days.
- Advertise bandwidth more than the median bandwidth of all the relays, or 2.0 MBps.
- Have a *weighted-fractional-uptime (WFU)*[1] of more than the median of all relays' WFU or 98% WFU.

4.2.3 Related Work

Elahi et al. [58] developed a framework called COGS to study guard selection schemes in Tor and evaluated the impact of churn, guard rotation, and the size of the guard list on clients' anonymity. Their results show that guard rotation exposes the users to more guards, increasing the chance of picking malicious guards. On the other hand, they find that

[1] WFU is the percentage of the time that relay has been up, adjusted with a decay of 5% per every 12 hours the relay is off.

guard rotation offers better load balancing on guards, better utilization of recently joined guards, and regaining the privacy of clients stuck using malicious guards. If Tor rotates the guards, they report, then larger guard lists lead to more compromises of anonymity; without guard rotation, larger guard lists lead to fewer compromises. They also find that larger guard lists lead to better, fairer performance.

Johnson et al. [46] evaluate the vulnerability of Tor to passive end-to-end correlation attacks from both relay- and network-level adversaries, with the settings used prior to 2014 of three guards rotated after 30 to 60 days. They define metrics that give us the probability of path compromise for a given user and the probability of time to the first compromise. They developed a path simulator that mimics the Tor client and implements multiple models of user activity, such as Web users and BitTorrent clients. They found that relay-level adversaries can maximize their resources by allocating more bandwidth to malicious guards than to malicious exits. Against an adversary running one 83.3 MBps guard relay and one 6.7 MBps exit relay, they find that 80% of users will be compromised within six months. Considering network-level adversaries, they find that an AS adversary can compromise 38% of Tor streams in three months, and a IXP adversary can compromise 20% of streams in the same period. We implement our guard selection scheme in their path simulator to analyze the security of our design at the network level.

Based on the results from Elahi et al. and Johnson et al., Dingledine et al. [57] conclude that the guard selection mechanism in use prior to 2014 harmed users' security. Instead of using three guards for 30 to 60 days, they proposed using a single guard for nine to ten months. They note that, based on Elahi et al's findings, this proposal will provide stronger anonymity but suffers from poor performance and poor load balancing, as the newly joined guards will be underutilized. To fix these flaws, Dingledine et al. suggest raising the bandwidth bar in assigning guard flags from 250 KBps to 2MBps and having

underutilized guards act as middle nodes. These changes were implemented in Tor and are still in effect as of the time of writing.

Dingledine el at. also suggested the idea of guard sets [57], which Hayes and Danezis [14] then expanded into a full proposal and evaluation. The Hayes and Danezis design puts the guards into sets based on their bandwidths, such that each set has approximately the same bandwidth. Although this approach is intuitive, we show in the following section how it leads to vulnerabilities that we then address in our proposed design.

4.3 Attacking Guard Sets

With guard sets, the directory authorities put all the guards into sets and include this assignment in the consensus [57]. The client randomly picks a guard set to use for a long period of time and then picks the guard for each circuit randomly from the selected set. The main advantage of this scheme is that it puts all the clients using a given guard set into one anonymity set, such that a guard fingerprinting attack could only identify one as a member of the set.

4.3.1 Hayes and Danezis Design

Hayes and Danezis performed the first detailed study of the guard set idea, and they propose algorithms for how to build guard sets, assign users to those sets, and maintain the sets as the Tor network changes [14]. To ensure load balancing, their proposal uses bandwidth as the main criteria to build the sets. In particular, it first uses the bandwidth values from the consensus to generate *bandwidth quanta*, where each quantum represents a block of bandwidth from a single guard node. Using an empirically selected threshold of 40 MBps, a guard's bandwidth is divided into multiple quanta such that each quantum is above the threshold. A guard that has bandwidth BW generates $\lfloor \frac{BW}{40} \rfloor$ quanta, meaning that guards with less than 80 MBps bandwidth make up just one quantum. For example, if

we have guards with bandwidths of 10, 70, and 90 MBps, we get quanta 10, 70, 45, and 45 MBps. The bandwidth quanta then are sorted from largest to smallest.

To build guard sets, the algorithm goes through the sorted list of quanta and moves one quantum at a time from the head of the list to the current set until the total bandwidth of the set reaches the threshold of 40 MBps. Then the current set is added to the list of sets, and a new set is started. If the leftover bandwidth quanta in the sorted list make up less than 40 MBps, they are not used to build sets and do not contribute to any guard sets. The goal of sorting the quanta list is to put guard nodes with similar bandwidth in the same set, and it also forces an attacker with many low-bandwidth guards into fewer sets with similar bandwidths instead of being spread out into sets with mixed bandwidths.

Over time, the total guard bandwidth in Tor fluctuates, as some new guards join the network and others go offline. These changes affect the bandwidth of guard sets and the available bandwidth quanta. To address this, the strategy of Hayes and Danezis is to first repair *damaged* guard sets with bandwidth of less than 20 MBps. When repairing a given damaged set, the algorithm finds the leftover bandwidth quanta that fall between between 50% to 100% of the maximum guard bandwidth of the set. This list of quanta is called the *candidate list* of the set. Quanta from the candidate list are added to the set one by one until the set's total bandwidth exceeds 40 MBps. Once all damaged guard sets are repaired, the algorithm builds a new guard set from any remaining leftover quanta if their combined bandwidth is more than 40 MBps.

4.3.1.1 Vulnerabilities

Using bandwidth similarity to repair the sets opens a door for the attacker. The primary issue is that the attacker can identify guard sets that are close to breaking, i.e. around 20 MBps, and then add guards or tune his guards' bandwidths to have similar bandwidths. For example, if the vulnerable guard set has guards with bandwidths of 3–4 MBps, and the

attacker has an unused guard with 5 MBps bandwidth, he can set it to offer a maximum of 3.5 MBps to Tor. Once these sets break, the repair algorithm will include the adversary's guards to be in the candidate list, increasing its chances of joining a particular set. Additionally, the attacker can create new compromised guard sets by adding his guards to the network or tuning their bandwidth so that the total bandwidth of leftover quanta and his guards is above 40 MBps, causing the algorithm to build a new set.

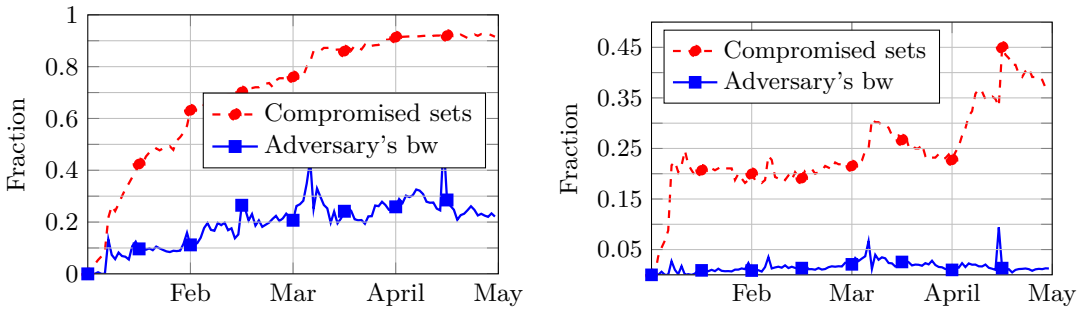
Another issue is that a guard set is not considered in need of repair if its bandwidth is at least 20 MBps. This means that as soon as the adversary joins a set, it can reduce the allocated guard bandwidth to the least possible value for being a guard, which can be as low as 2 MBps as long as the set's total bandwidth remains above 20 MBps. This can save the adversary's resources. Also, if an attacker gets one of his guards into a set, it will remain in that set forever even if all the other guards in the set are gone. This allows the attacker to retain a full guard sets' allocation of users while only using half of the bandwidth needed for building a new set.

4.3.2 Evaluation

We investigate the impact of these vulnerabilities on the adversary's ability to infiltrate guard sets and compromise Tor users. Jamie Hayes provided us with his implementation of the Hayes-Danezis algorithms for guard sets. We used their implementation and exploited the possible vulnerabilities in simulation.

4.3.2.1 Attacker Model

The goal of the attacker is to get into as many guard sets as possible and thereby compromise a large fraction of users. We run the Hayes-Danezis algorithm using consensus documents from January to May 2013, the same time period used by Hayes and Danezis for consistency. Like Hayes and Danezis [14], in all of our simulations we used the first



(a) High-bandwidth adversary

(b) Low-bandwidth adversary

Figure 4.2. **Hayes-Danezis:** The fraction of compromised sets and fraction of the adversary's bandwidth to the total guard bandwidth..

consensus documents of each day, instead of hourly consensus updates, to allow for longer studies. We allow the attacker to add his guard relays to the network in the second day of simulation after the guard sets are formed, and the attacker aims to both get into new sets and into previously built sets.

We assume that the attacker has access to the guard relays' bandwidths, which he can obtain from the consensus and refine if necessary by periodic measurements. The attacker also keeps track of the assignment of guards to guard sets, which is also available in the consensus. The attacker uses this information to follow the sets' bandwidth and detect which sets are about to break. If a set is about to break and the set is already compromised, the attacker tunes his bandwidth to keep the set alive by keeping the set's bandwidth above 20 MBps. Otherwise, if a set is broken and not yet compromised, the attacker adds new guards to the network with bandwidths tuned to get added to the set's candidate list. Instead of adding new guard relays to the network, the attacker can reuse his unused guard relays and tune their bandwidth. If the attacker observes new sets are forming from leftover guards, the attacker adds new guards or re-uses his guards in the leftover quanta and tunes their bandwidth to get to the list of candidates for the new set. Because the quantum in the

candidate list of a set are in descending order, The attacker only needs to tune his guard relay bandwidth slightly higher than the last bandwidth in the list which fixes the set.

In our simulations, we assume the attacker can add the new guards the same day he needs them. Note that the new sets are created and broken sets are repaired whenever there is enough bandwidth or a set is broken, but the changes are announced in the next consensus file. Also, the attacker's relays must be assigned the guard flag by the authorities. To do this, the attacker can run some relays that have all the criteria to get the guard flag (as listed in Section 4.2.2) except for one. For example, the attacker can run relays with high uptime but with bandwidth less than the minimum bandwidth required (currently 2 MBps), which saves his bandwidth while waiting for a set to break. When the attacker needs a new guard, he just needs to increase the bandwidth of the relay to get the Guard flag. Another technique would be for the attacker, who typically would run exit nodes to perform end-to-end correlation attacks together with his guards, to switch one of his exits to being a guard. This is easily done by first having a high-uptime relay with an exit policy, which will cause it to have the Exit-Guard flag, but it will be used exclusively as an exit due to exits being the bandwidth bottleneck in Tor. Then, to switch it to a guard, the attacker simply removes the exit policy.

We used two adversary models, a *high-bandwidth adversary* who controls about 25% of Tor's bandwidth and a *low-bandwidth adversary* who controls about 1%. The actual bandwidths being used by the adversary vary over time, as shown by the solid lines in Fig. 4.2. For both models, when the adversary gets into a guard set, he reduces his relay's bandwidth as long as the total bandwidth of the set remains above 20 MBps. In the low-bandwidth adversary model, the adversary leaves a set if he is the set's main resource provider, which we define as providing more than 90% of the set's bandwidth. In compromising the guard sets, the adversary needs to have only one guard in the set to compromise

all the clients attached to that set. If more than one adversary guard is assigned to a set, the adversary will pull one of his guards from that set to inject it into the other set.

We assume that the attacker knows exactly when guard relays will break. In reality, however, the attacker will spend additional resources waiting for guard sets that are close to 20MBps to break. Also, the attacker may miss some guard sets that have sudden large drops in bandwidth. Thus, the results of our experiment represent an upper bound of compromised sets for a given amount of attacker resources.

4.3.2.2 Results

Figure 4.2 shows the upper bound of the fraction of compromised sets and the ratio of the adversary's bandwidth to the total guard bandwidth in Tor for the first half of 2013. As shown in Figure 4.2(a), a high-bandwidth adversary who owns around 25% of the total guard bandwidth in the Tor network can compromise more than 90% of all the guard sets (and thus be a guard for over 90% of all the clients) in just four months. Figure 4.2(b) shows that a low bandwidth adversary with only 1% of total guard bandwidth can compromise around 40% of all the guard sets in just four months. Large fluctuations in the graph are due to large drops in guard bandwidth that occurred in early March and mid-April. Our attacks shows that against the prior guard set design, an adversary with modest resources can endanger the security of a large number of users.

4.4 Design

To mitigate the threat of an adversary compromising a significant number of guard sets, we seek an approach that is more resilient in the face of frequently changing guard sets. In particular, the method should make it harder for an attacker to join a targeted guard set in need of repair and allow clients to keep as much of their anonymity sets as possible, even

when guard sets break. To this end, we propose to take advantage of the relative stability of the underlying Internet topology by linking sets to customer cones. In this section, we first explain our motivation for the design, and then we describe how guards are grouped into sets using customer cones and how guard sets are assigned to clients.

4.4.1 Motivation

To prevent the attacks we describe against the guard set design of Hayes and Danezis in the previous section, we need to prevent an attacker from easily joining arbitrary guard sets. First, as Hayes and Danezis also argue [14], we should maintain a hierarchy of guard sets, represented as a tree. This hierarchy dictates that when guard sets are deleted, there is a pre-defined backup guard set for the users of the old set to join. This keeps users together as much as possible, maintaining their anonymity sets. Beyond the Hayes and Danezis proposal, we also would have guards remain in the same place in the hierarchy as much as possible. When a guard set is deleted, the remaining guards should stay in the same general area in the hierarchy, i.e. with siblings in the tree. Also, new guard sets should only be constructed from guards in the same subtree. This prevents guards from attempting to move from one part of the tree to another.

The other major requirement of our approach is that the attacker must not be able to place new nodes into arbitrary locations in the hierarchy. A simple approach would be to use a cryptographic hash of the node's IP address as an identifier, much like in a DHT. Unfortunately, an attacker with even a fairly small range of IP addresses to use could pick a number of different locations in the hierarchy by computing their hash values in advance. If the directory server were to pick the locations of new guards randomly, the attacker could add and remove nodes until the location suited his needs.

In our design, a guard's place in the hierarchy and guard set assignment is based on the guard's network location, meaning the AS it is in and that AS's corresponding place

in the customer cones of the Internet. For an adversary with high bandwidth capacity and a range of IP addresses, but only a few network locations, this would substantially limit the number of guard sets he can join and the number of users that he can compromise. To fully overcome this, the attacker would need to be able to place guards into *arbitrary* network locations that have guards. We argue that this attacker model is unlikely in practice. A botnet-based attacker, for example, will likely face challenges with the stability and bandwidth requirements for guards. Even if enough stable bots can be found, the bot locations (such as consumer ISPs) may not correlate well with the locations of Tor guard nodes (which include professional hosting services like OVH), and this further limits the guard sets he can join.

Overview. In the rest of this section, we describe our proposed hierarchy. The hierarchy consists of three levels: 1) *Root Sets*, 2) *Branch Sets*, and 3) *Guard Sets*. A Root Set is a customer cone of a root AS that contains guard ASes. Root Sets are broken into Branch Sets and then further into Guard Sets. Branch Sets represent smaller customer cones within the Root Set, in which all guard ASes have the same provider. Finally, Guard Sets are formed by selecting all guards within a Branch Set and grouping the guards such that the number of ASes within a Guard Set are minimized and the guard bandwidth is above a threshold. Below, we describe each part of the system in detail.

4.4.2 *Root Sets*

To form Root Sets, we first build a *root set list* – a list of ASes sorted based on customer cone size in ascending order. Initially, the *root set list* contains all ASes with one or more guards (*guard ASes*), and each guard AS is considered as a Root Set. Consider Figure 4.1 as an example for this section, which means that the list would be something like {AS13, AS14, AS5, AS6, AS7,AS9, AS10, AS11, AS12}, assuming that only the

leaf ASes have guards. Then we choose the Root Set with the smallest customer cone size, say AS13. We follow all c2p links to discover all providers for this Root Set that are also providers to at least one other Root Set in the list, e.g. AS8, AS3, and AS1. Among these providers, we select the provider with the smallest customer cone size, e.g. AS8. This provider becomes the new Root Set and is added to the *root set list*, while the Root Sets that are in the customer cone of this provider (AS13 and AS14) are removed from the *root set list*. This process is repeated until all Root Sets in the *root set list* contain guard bandwidths more than a bandwidth threshold τ_{up} or the number of Root Sets in the list have decreased below a threshold N.

Updating Root Sets As new guard ASes join the network, the algorithm first checks whether the new guard ASes are in the customer cone of an existing Root Set. If they are in an existing Root Set's customer cone, they are added to that Root Set. If there are still some guard ASes that are not in any of the Root Sets' customer cones, they themselves are considered as Root Sets. Then the above algorithm is run to reform the Root Sets.

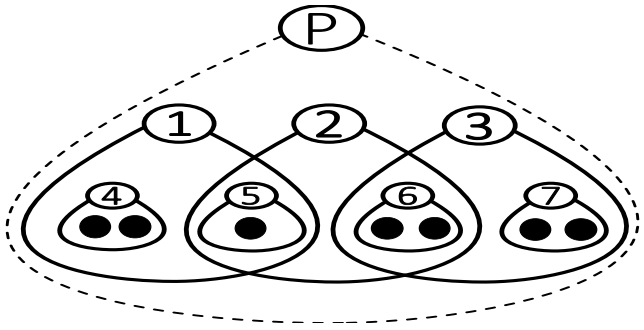
4.4.3 *Branch Sets*

Root Sets often represent large customer cones and many guards. To better isolate groups of guards from each other and make it harder for a malicious guard to move into targeted guard sets, we break each Root Set into Branch sets. In building Branch Sets, the goal is to place guard ASes that are close together in the AS relationship graph into the same Branch Set. To this end, we first identify all customer cones within the Root Set's customer cone in which the guard bandwidth reaches the threshold τ_{up}.[2] Note that some cones will be contained within other, larger cones, and there can be overlaps between cones. Among all the possible customer cones, we should pick cones such that their inter-

[2]This is the same threshold as used to make the Root Sets.

section with respect to guard ASes is empty ($A \cap B = \emptyset$). There may be many possible combinations of customer cones that are independent in this way. Since our goal is to have smaller customer cones to make it harder for an attacker to join a targeted Branch Set, we pick the combination that has the maximum number of independent cones. Each of these independent cones will be a Branch Set within the given Root Set. At the end, we place all guard ASes that do not meet the requirements for building Branch Set into one additional set.

Figure 4.3 shows an example of Branch Set creation for Root Set P. There are seven customer cones with sufficient bandwidth, but there are overlaps between some of them. The possible combinations of independent cones are $\{1, 3\}$, $\{3, 4, 5\}$, $\{2, 4, 7\}$, $\{1, 6, 7\}$, and $\{4, 5, 6, 7\}$. Among these combinations, the algorithm picks $\{4, 5, 6, 7\}$ because it has the maximum number of independent cones. This means that Root Set P has four Branch Sets.



(a)

Figure 4.3. **Branch Set creation.** The dashed line shows Root Set P's customer cone, the black circles are guard ASes, and the solid lines show the customer cones with bandwidth τ_{up} or greater..

Updating Branch Sets. Guard bandwidth fluctuates over time, causing some guard ASes to be dropped from the Branch Set and others added, which requires periodic updates. Our

algorithm to update the Branch Set first checks which of the new guard ASes are in our current Branch Set' customer cones. If a Branch Set's bandwidth is below threshold τ_{down}, it dismantles the Branch Set and releases its guard ASes. The algorithm is then run again to build Branch Set from previous Branch Set, new guard ASes, and released guard ASes.

4.4.4 Guard Sets

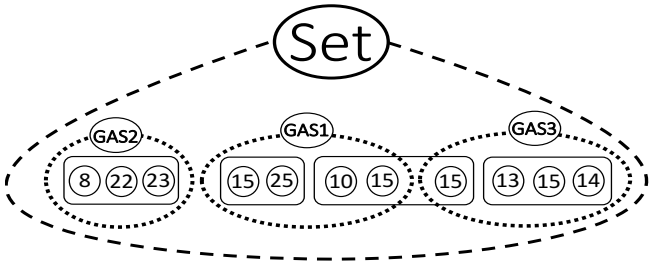
Once we have Branch Sets, we can break them up further into the Guard Sets. We first randomly shuffle the guard ASes in the Branch Set. Then we add one guard at a time from the same AS to the current Guard Set until the Guard Set's bandwidth reaches the threshold τ_{up}.[3] If we use all the guards in an AS, we continue adding guards from the next guard AS.

Figure 4.4 shows an example of Guard Set creation. The Branch Set's customer cone includes three guard ASes, GAS1, GAS2, and GAS3, and four Guard Sets are formed. Note that a Guard Set can include all of the guards in an AS (such as in GAS2), some of the guards in an AS (such as the leftmost Guard Set in GAS1), or guards from multiple ASes (the two rightmost Guard Set).

Updating Guard Sets Guard Sets will need to be updated over time due to the leaving and joining of guard ASes and changes in bandwidth. If a Guard Set's bandwidth falls below the threshold, τ_{down}[4], we need to repair that Guard Set to ensure load balancing. To repair a low-bandwidth Guard Set, we add new guards to it until the bandwidth reaches at least 40 MBps. We first try to add new guards that are in the same AS, and then add from other guard ASes in the Branch Set. If there are still some unused new guards in the Branch Set, we try to build new Guard Sets out of them.

[3] Again, this is the same threshold as used for Root Sets.

[4] The same threshold as for updating Branch Sets



(a)

Figure 4.4. **Guard Set creation.** Dashed ovals represent guard ASes (GAS), rectangles represent Guard Sets, and circles represent guards. Numbers inside the circles are the guards' bandwidths (MBps)..

4.4.5 Assigning Clients to the Guard Sets

A newly-joined client selects first a Root Set, then a Branch Set from among the Branch Sets in her Root Set, and finally a Guard Set from among the Guard Sets in her Branch Set. Each of these selections is random, weighted proportionally by bandwidth. To create a circuit, the client picks one of the guards in her Guard Set as the entry relay. The selection of guards from a Guard Set can be weighted in favor of bandwidth or can be done uniformly at random.

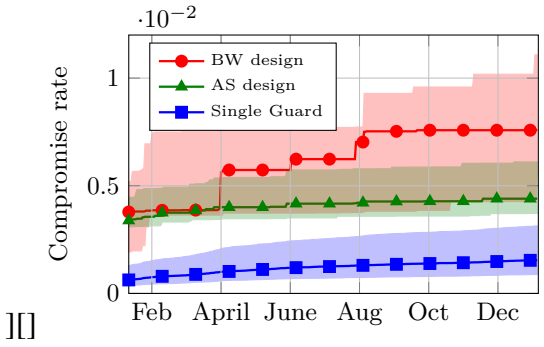
As time passes, some guards leave the network, and this causes some Guard Sets, Branch Sets, or even Root Sets to be no longer available. If the client's Guard Set has been dismantled, the client will select another Guard Set under her Branch Set. Similar recovery methods are available for Branch Sets and Root Sets. In the worst case, if her Root Set is gone, the client acts like a newly-joined client.

4.5 Evaluations

In this section we evaluate different aspects of our guard set design. We start by analyzing the security of the proposed guard set design in the presence of relay- and network-

level adversaries. Then we monitor guard set changes over time with respect to the number of guard sets, guard set bandwidth, and client anonymity sets. In our evaluation, we set bandwidth thresholds $\tau_{up} = 40$ MBps and $\tau_{down} = 20$ MBps – the same as the thresholds used by Hayes and Danezis [14]. We set the number of Root Sets to $N = 50$ to maintain enough Root Sets and not to be merged into only top tier ASes. For our data set, we use consensus documents from January 2015 to December 2015 from Tor Metrics [48]. Following Hayes and Danezis, during the entire evaluation we do not rotate the guards. Additionally, we compare our results to Hayes and Danezis's design [14], which we refer to as "*BW* design." We call our design "*AS* design."

4.5.1 Security Evaluation



][]

Figure 4.5. *Low-resource* adversary: the solid lines show the median of compromised clients and the colored bands shows the area between the first and third quantiles..

Tor is known to be vulnerable to traffic correlation attacks [7, 11, 64, 46, 44]. An adversary who observes both entry and exit traffic can use the timing of packets to link clients to their destinations. Observing both sides of Tor traffic can happen at the relay level or the network level. At the relay level, an adversary running guard and exit nodes in the Tor network may deanonymize clients whose circuits traverse the adversary's guard and exit relays. We examine two relay-level attack scenarios, a non-targeted relay-level

adversary and a targeted relay-level adversary. At the network level, the adversary controls some part of the network, such as an Autonomous System (AS) or Internet Exchange Point (IXP), and can thus observe huge amounts of traffic, including entry and exit traffic in Tor. In this section, we examine the security of our guard set design against both relay-level and network-level adversaries.

Security Claims. In this section, we seek to demonstrate the following:

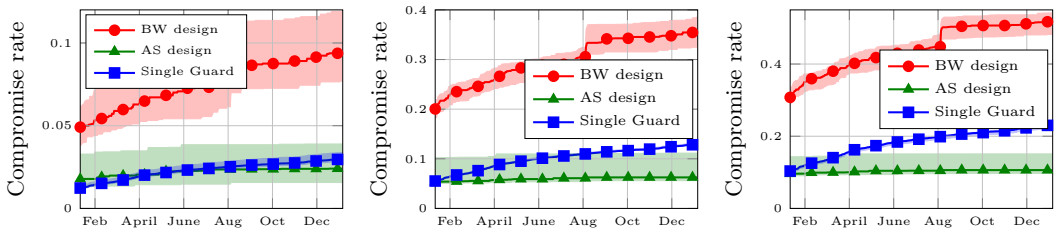
1. The compromise rate is lower for *AS* design compared to *BW* design for a variety of relay-level adversaries with varying resource levels and for both non-targeted and targeted attacks.
2. The vulnerable stream rate is approximately the same as both Tor and *BW* design against network-level adversaries.
3. *AS* design is compatible with the DeNASA [59] AS-aware path selection algorithm, and the combined algorithms provide similar vulnerable stream rates as DeNASA against network-level adversaries.

4.5.1.1 Non-Targeted Relay-Level Adversaries

A non-targeted relay-level adversary adds guard nodes in the network to compromise guard sets. This adversary does not target any specific guard set or client; his goal is to compromise as many clients as possible, which means joining as many guard sets as possible.

The adversary model. If a guard set contains one compromised guard relay, we consider the entire guard set to be compromised; all clients using that guard set will be compromised because they eventually send traffic through the compromised guard. This follows the model of Hayes and Danezis [14]. We examine the relationship between the amount of guard bandwidth the attacker provides to the Tor network and his success rate in compro-

mising guard sets. For each of the guard selection strategies, *AS* design, *BW* design, and *Single Guard* (i.e. Tor), we assume that the adversary runs some guard relays such that their total bandwidth adds up to 1%, 5%, or 10% of the total guard bandwidth of Tor for different experiments.



(a) 1% of guard bandwidth (b) 5% of guard bandwidth (c) 10% of guard bandwidth

Figure 4.6. **High-resource centralized adversary.** Compromise rates for varying fractions of total guard bandwidth owned by the adversary..

Because *AS* design uses both bandwidth and AS relationships, the adversary's network (AS) matters as well. Therefore, we analyze the security of our guard set design under three attack strategies: a *low-resource* adversary, a *high-resource centralized* adversary, and a *botnet* adversary. In Tor, a client is considered compromised if it chooses a malicious guard. We note that this is not completely fair to the guard set designs, since a single guard in Tor can compromise more of the user's traffic than any one member of a guard set in which each guard is picked only part of the time. We allow the adversary to inject malicious relays at the beginning of the simulation before the 500,000 users that we simulate start picking their guards. Unlike the model used by Hayes and Danezis [14], we assume that the adversary's guard relays remain up and available during the entire simulation, which gives the adversary an advantage.

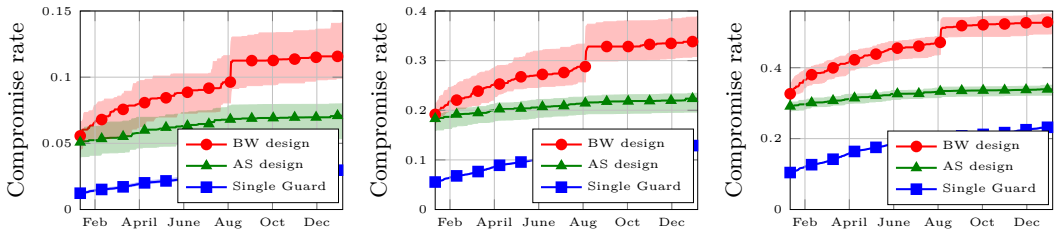
Low-resource adversary. In this attack strategy, we assume that the adversary injects only a single guard relay in the network. We randomly choose an AS for this malicious guard

and select its IP address randomly from the IP range of the selected AS. We also randomly select a bandwidth value from Tor guards' bandwidths in the consensus document. This malicious guard is added to the network, and the simulation is run for the year of consensus files. We repeat the simulation 50 times, with a new malicious guard each time.

Figure 4.5 shows the median fraction of compromised clients over 50 simulations. At the beginning of 2015, the compromise rate of *AS* design is statistically similar to *BW* design. The compromise rates of both *AS* design and *BW* design are greater than *Single Guard*, because it is assumed that a single malicious guard within a guard set compromises all clients who choose that guard set.

Over time, the compromise rate in *BW* design grows substantially from 0.036% to 0.076%, as the malicious guard moves into different guard sets. On the other other hand, *AS* design's compromise rate only grows from 0.032% to 0.044%. We note that the variance in these results for low-resource adversaries is high, as seen by the wide quartile bands, but the trends are consistent. The growth of the compromise rate in *AS* design is 37%, much smaller than the 110% growth in *BW* design. The reason for this is that in *AS* design, malicious guards are quarantined inside a Branch Set within a Root Set. By design, the malicious guard cannot infiltrate guard sets that are outside of the malicious guard's Branch Sets. In contrast, in *BW* design, when the bandwidth in the network changes or the malicious guard's bandwidth changes, the guard sets change and the malicious guard moves from one guard set to another and compromises additional guard sets over time.

High-resource centralized adversary. In our model, the high-bandwidth adversary owns some relays such that its total bandwidth is a considerable fraction of the network's total bandwidth. We assume that the adversary is *centralized*, meaning that it injects all malicious guards into a single AS. We select at random one guard AS in which to add the malicious guard relays, and we select their bandwidths randomly from live Tor guard



(a) 1% of guard bandwidth (b) 5% of guard bandwidth (c) 10% of guard bandwidth

Figure 4.7. **Botnet adversary.** Compromise rates for varying fractions of total guard bandwidth owned by the adversary..

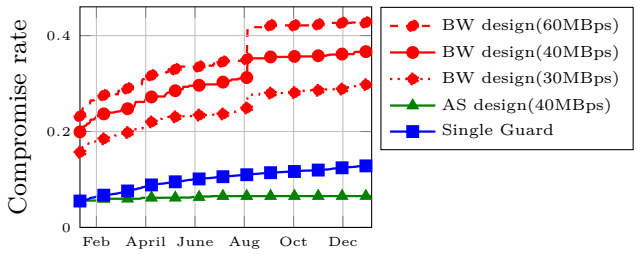
bandwidths. Such relays are added to the Tor network until the target attacker bandwidth is reached. The simulation is run 50 times, with a new guard AS and new malicious relays each time.

Figure 4.6 shows the fraction of compromised clients for three different guard selection schemes and three different adversarial bandwidth assumptions. As the adversary's bandwidth increases, the fraction of compromised clients increases for all three schemes. *BW* design has the largest fraction of compromised clients compared to the other two schemes. In *BW* design, we observed a significant increase in compromises in August 2015. We found that this was due to the large drop in Tor guard bandwidth in August 2015. This triggered a significant churn in guard sets for *BW* design that allowed malicious guards to infiltrate more guard sets. This suggests that bandwidth changes in the Tor network have a negative impact on security for *BW* design due to its vulnerabilities.

After one year of simulation time, we observed significantly higher growth in the fraction of compromised clients for *Single Guard* compared to *AS* design for all three adversarial models. For 1%, 5%, and 10% simulated adversarial bandwidth, the fraction of compromised clients for *Single Guard* increased by 180%, 168%, and 157% over one year, respectively. The fraction of compromised clients for *AS* design remained almost constant over one year. These results support our assertion that *AS* design successfully constrains

the adversary's guards to guard sets within the Branch Set and Root Set. Moreover, clients do not rotate guard sets unless their guard set is broken up. Even then, the client will choose another guard set within its Branch Set and Root Set. These characteristics allow the *AS* design to keep the compromise rate low over time.

We also explore how our results change for different bandwidth thresholds τ_{up} and τ_{down} in both *AS* design and *BW* design. Figure 4.8 shows the compromise rates for three different values of τ_{up}, while keeping τ_{down} = $\tau_{up}/2$. The compromise rate of *BW* design increases significantly as τ_{up} increases. On the last day of simulation, for example, the compromise rate for $\tau_{up} = 60MBps$ ($\tau_{down} = 30MBps$) is 0.42, a 44% increase over that of $\tau_{up} = 30MBps$ (0.29). The guard sets in *BW* design are more prone to breaking for larger values of τ_{up} and τ_{down}, which causes more guard rotation and an increased rate of compromise. For *AS* design, we observe only small changes in the compromise rate for changing bandwidth thresholds. For $\tau_{up} = 60MBps$ ($\tau_{down} = 30MBps$), the compromise rate goes from 0.056 on the first day to 0.068 on the last day of simulation, where the latter is only a 13% increase over the compromise rate when $\tau_{up} = 30MBps$ (0.06). This is due to the use of network location rather than bandwidth as the key criteria for managing sets in *AS* design. Note that for clarity, due to how close the results are, Figure 4.8 only shows the results for $\tau_{up} = 40MBps$.



(a)

Figure 4.8. **High-resource centralized adversary.** Compromise rates as τ_{up} varies, $\tau_{down} = \tau_{up}/2$, and the adversary controls 5% of guard bandwidth..

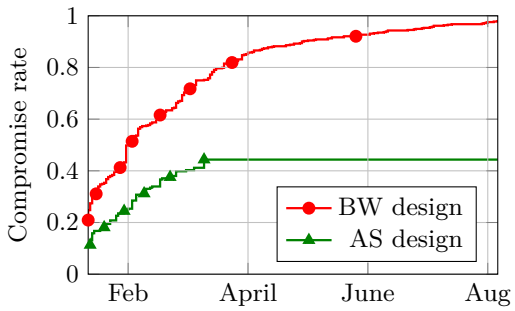
Botnet adversary. This adversary is similar to the high-resource centralized adversary, except the adversary injects his guard relays from different guard ASes instead of one guard AS. For each simulation, the adversary adds malicious guards to the network from different guard Ases, selected randomly from all guard ASes, until the desired bandwidth for adversary is reached. We also repeat this simulation 50 times.

Figure 4.7 shows the fraction of compromised clients for all three guard selection schemes in the presence of a *botnet* adversary with different bandwidth fractions. We see that the results for BW design and *Single Guard* do not change compared to the *high-resource centralized* adversary (Figure 4.6) because these two methods work with bandwidth in either grouping guards or client assignments and do not use the guard ASes. Because the adversary's relays are in different ASes, however, he can compromise many more guard sets in the AS design. As we see in the figures, *AS* design's compromise rate in this attack is higher than its compromise rate in Figure 4.6. Nevertheless, *BW* design's compromise rate rises much faster than *AS* design's, which is almost constant over time. This indicates that, although this adversary can compromise many sets, the AS design is good at stopping the propagation of the adversary's impact on the network. *Single Guard* has a compromise rate less than the other two guard set designs, which is a trade-off against the smaller anonymity sets provided by *Single Guard*. The growth of compromise rate over a year against *Single Guard* is 194%, 168%, and 155% for 1%, 5%, and 10% simulated adversarial bandwidth, respectively. These compromise rate growths are much higher than 62%, 17% and 22% growth seen against *AS* design for 1%, 5%, and 10% simulated adversarial bandwidth, respectively.

4.5.1.2 Targeted Attacks

The other case we examine is when the adversary targets a specific client, adding guard relays to Tor with the goal of getting added to the client's guard set. The way *AS*

design builds and repairs the sets makes it harder for the adversary to get into the targeted client's guard set.



(a)

Figure 4.9. **Targeted attack.** CDF of time to compromise..

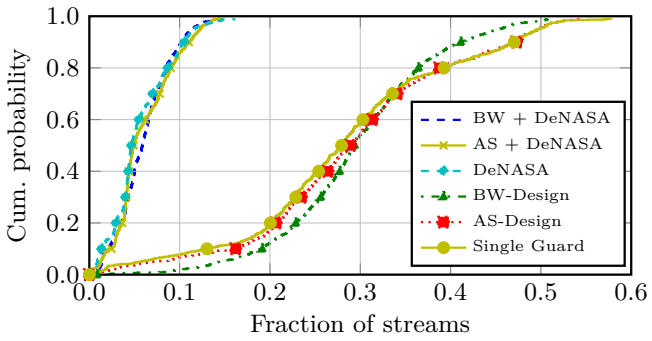
The adversary model. The attacker controls one or more exit nodes and profiles a particular user of interest based on her exit traffic. We assume that this adversary can identify the client's guard set. Given that all clients know the assignment of guards to guard sets, an adversary simply needs to run one Tor client to get this mapping. It is more challenging to learn the assignment of the client of interest to her guard set. A variety of attacks, however, reveal the clients' guard relays [65, 12, 11, 66, 54], and we assume that the attacker uses one of these attacks successfully to identify one of the guards and the corresponding guard set. Then, the goal of the attacker is to get into this guard set.

We simulated the attack for *BW* design and *AS* design. In the simulations, we select a target client and assign a guard set to this target on the first day. Then the adversary monitors the network, measures the guard bandwidths, and waits until the target's guard set is about to break. We assume that the attacker can add some guard relays whenever he chooses—we discuss the validity of this assumption in Section 4.3.

Against *AS* design, when the target guard set is broken, the attacker selects one of the guard ASes used in that guard set as the AS from which he injects his guard relays. The

attacker finds all the broken guard sets in which this AS belongs and computes the total amount of bandwidth needed to fix all of those broken guard sets. Then the attacker adds guard relays from the chosen AS to the network until they reach the required bandwidth. The attack against *BW* design proceeds as described in Section 4.3.

We simulated the targeted attack for the course of one year from January 2015 to December 2015. On the first day, January 1, we added a target client and assigned a guard set to her based on the given guard set assignment algorithm. Once the target guard set is determined, we wait until the target guard set is broken. We then add the adversarial relays to the network and tune their bandwidth based on the bandwidth needed to get into that set. We continue monitoring the network, tuning the adversary's bandwidth and adding more guard relays as needed until the target is compromised. When the target is compromised, the adversary tunes his bandwidth to keep the guard set alive, i.e. the target set's bandwidth should not be less than τ_{down}. We repeat this process for 500 targeted clients.



(a)

Figure 4.10. **AS level adversary.** CDF of vulnerable stream rates..

Results. Figure 4.9 shows the cumulative fraction of days it takes for the targeted clients to be compromised. With *BW* design, almost all the targeted clients (98%) were compromised within the year, and 50% of the targets were compromised within one month. *AS* design

protects clients better from the targeted attack, as just 44% of targets were compromised within the year. The reason for this is that if the target guard set is broken, it is repaired by guard relays from ASes in the same Branch Set. This limits the chances for the adversarial AS to be picked to repair the set. Overall, we find that clients in *AS* design are safer than in *BW* design.

4.5.1.3 AS-Level Adversaries

An adversary may be able to monitor network traffic on one or more ASes or IXPs on the Internet and observe both sides of Tor circuits to link users with their destinations. In this study, we consider a stream to be *vulnerable* if both the entry and exit sides of the traffic traverse the same AS. To examine the security of our design at the AS level, we implemented the guard selection schemes in TorPS [46] and generated streams from a set of clients to a set of destinations. Then we found the AS paths on both the forward and reverse connections from client to guard, and exit to destination [44]. We had 6,000 clients connecting from 30 client ASes distributed over the top countries using Tor. To choose the 30 ASes, we first pick a country for a user based on the distribution of users from the top countries according to Tor Metrics [52]. We then check whether this country has an AS in top client ASes list given by Edman et al [67]. If so, we pick that AS and add it to our client AS list, and remove that AS from the top client ASes list. Otherwise, we randomly select an AS from that country and add it to our client AS list. We keep selecting countries based on the distribution of directly connected clients distribution until we have 30 client ASes. Over one month of simulation time (Feb. 2015), these clients generated 8 million streams for each guard selection mechanism.

Figure 4.10 shows the CDF of vulnerable stream rates for clients using *AS* design, *BW* design, and *Single Guard*. As shown, the fraction of vulnerable streams is almost the same for the three guard selection mechanisms with a median of 28%. Thus, *AS* design

appears to provide similar anonymity as *Single Guard* and *BW* design against an AS-level adversaries.

We combined the guard selection schemes with DeNASA [59], an AS-aware path selection algorithm. DeNASA avoids paths with *suspect* ASes, mainly Tier 1 ASes that appear frequently on the entry and exit sides of Tor traffic.

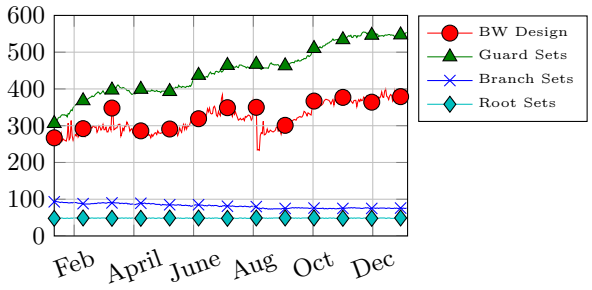
DeNASA Implementation DeNASA [59] is a recently proposed AS-aware path selection algorithm. It avoids paths with *suspect* ASes, mainly Tier 1 ASes that appear frequently on the entry and exit sides of Tor traffic. The main advantage of this approach is that it is destination-naive, which enables Tor to preemptively build circuits for performance reasons. The downside of DeNASA is that it is vulnerable to leakage about a clients AS across repeated connections [68].

We implemented the $g\&e$-select algorithm introduced by DeNASA. In combining DeNASA with guard set designs, the client first picks a guard set (or a guard relay in *Single Guard*) in such a way that is defined in each guard selection mechanism. If there is a suspect AS on the AS path between the client and any of the guards in the chosen guard set, the client drops that set and tries another guard set. Otherwise, she keeps the guard set. At the time of building a circuit, DeNASA picks a guard relay from the chosen guard set; then it picks an exit relay such that Tor picks, if the probability of appearing suspect ASes existing on the entry path (the AS path between the client and and the guard relay) is less than a threshold, the chosen exit is acceptable. Otherwise, it tries another exit relay. We set the probability threshold to 0.1 in our simulations. .

The suspect ASes for the entry side, the suspect ASes which appear more frequently on the path between the clients and guard relays, are :{ 1299, 3356}. The suspect ASes for the exit side, the suspect ASes which appear more frequently on the path between the exit relays and destinations, are :{ 1299, 3356, 6939, 174, 2914, 3257, 9002, 6453}.

The probability table is the input to DeNASA algorithm. The rows in the table are exit ASes, which have exit relays, and columns are the suspect ASes. In the table, each value P_{ij} represents the probability of appearing suspect AS *j* on the AS paths between exit AS *i* and the possible destinations. We considered the possible destinations all the destinations visited by TorPS *typical* user model.

As shown in Figure 4.10, after combining the guard selection mechanisms with DeNASA, the median vulnerable stream rate for both guard set designs and *Single Guard* dropped 80% (from 0.28 to 0.05). Thus, we believe that *AS* design is compatible with DeNASA for protecting against AS-level adversaries.



(a)

Figure 4.11. Counts of *AS* Root Sets, Branch Sets, and Guard Sets and *BW* sets..

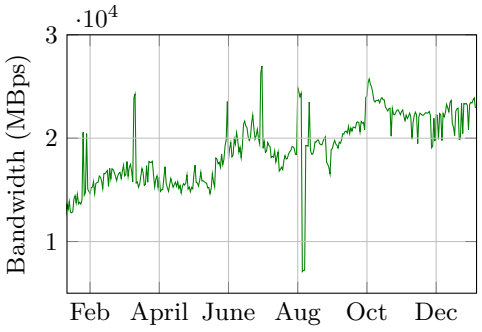
4.5.2 System Evaluation

We now examine dynamics in the number of sets, sets' bandwidths, and anonymity sets.

System Claims. In this section, we seek to demonstrate the following:

1. The hierarchy derived from the customer cones is stable over time, with moderate changes to Branch Sets and Guard Sets.

2. The anonymity sets of users are significantly higher for *AS* design over *Single Guard* and approximately the same as *BW* design.
3. Bandwidth is distributed sufficiently evenly between guard sets in *AS* design to not create bottlenecks at the guard or waste significant bandwidth.
4. Network performance in Tor is approximately the same for *AS* design compared with *Single Guard*.



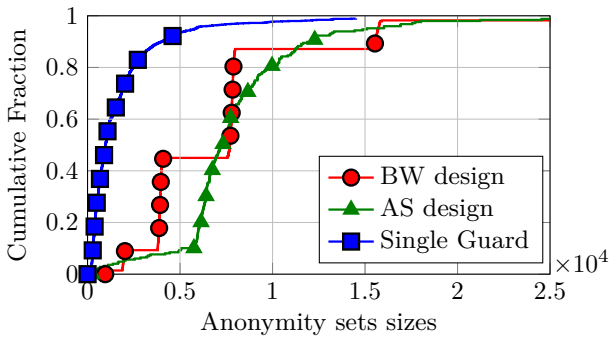
(a)

Figure 4.12. Daily guard bandwidth throughout 2015..

Guard Sets. Our guard set design includes Root Sets, Branch Sets, and Guard Sets. Figure 4.11 shows the number of each of these elements over the year 2015. The number of Root Sets was 48 and did not change throughout the experiment.

The number of Branch Sets changed modestly over time, ranging from 76 to 92 with an average of 84. This shows that the customer cones that make up the Branch Sets do not change greatly over time. Guard Sets increased over time because they are built at the relay level and based on relay bandwidth. Figure 4.12 shows the daily guard bandwidth in the Tor network, and we observe three significant increases in bandwidth: in late February, late June, and early October. These increases in bandwidth correspond closely with the increase of guard sets in Figure 4.11. During our simulations, on average 15 guard sets in *AS* design

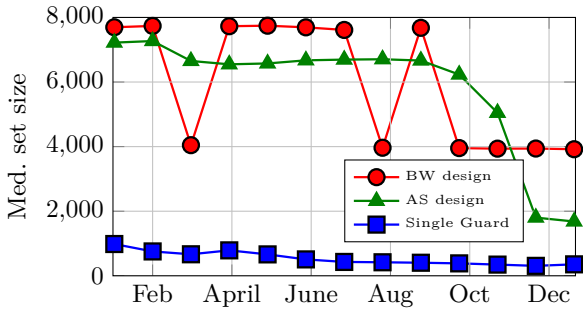
got repaired each day, compared with seven guard sets in *BW* design. *AS* design often adds guards from broken sets to existing sets, and this counts as a repair of the existing set.



(a)

Figure 4.13. CDF of anonymity set sizes..

Anonymity Sets. We define an anonymity set as a set of clients that use the same set of guards. If the size of the anonymity set is small, then the threat of the guard fingerprinting and statistical disclosure attacks increase. To evaluate the anonymity set sizes in our design, we use our client assignment mechanism described in Section 4.4.5 to attach clients to guard sets. In this experiment, we model 2,000,000 clients, which is approximately the number of Tor daily users [48].



(a)

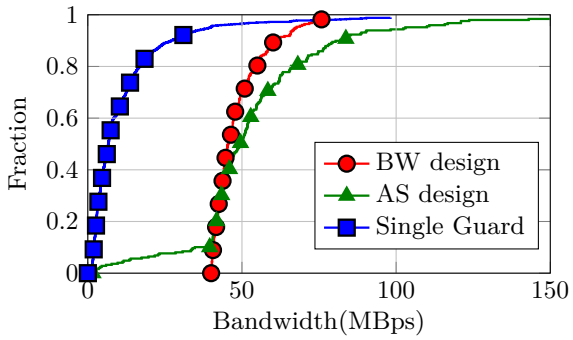
Figure 4.14. Median of anonymity set sizes over time..

We compare the anonymity set sizes in *AS* design with the ones in *BW* design and Tor (*Single Guard*). Figure 4.13 shows the empirical CDF of anonymity set sizes. The median anonymity set size in *AS* design (7300 clients) is almost the same as *BW* design (7700 clients), and both are far larger than in Tor currently (970 clients). Figure 4.14 depicts the changes in median anonymity set size over time. The median for both *BW* design and *AS* design decrease over time. Because we do not rotate guards and do not consider user churn in our simulations, new guard sets have few users and small anonymity set sizes. The sizes decrease particularly fast from September, which corresponds to when the number of guard ASes starts increasing most rapidly.

Set Bandwidth. To ensure network performance remains similar to Tor, we must ensure that guard bandwidth is distributed relatively evenly over guard sets. To do this, we test whether all guard sets have an accumulative bandwidth above a certain threshold.

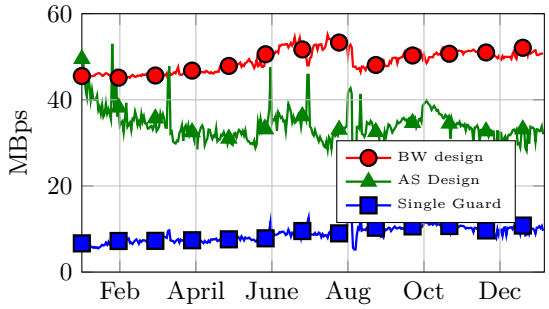
Figure 4.15 shows the CDF of bandwidths among the guard sets in *AS* design and *BW* design and among individual guard relays in *Single Guard*. The results for *AS* design and *BW* design are similar, though *AS* design has more high-bandwidth guard sets. This is because it does not break up a high-bandwidth guard into multiple sets the way that *BW* design does. Note that *AS* design retains good load balancing in this case by having clients pick their guard sets with a weight for bandwidth.

Figure 4.16 shows the median guard set bandwidth over time in *AS* design and *BW* design and the median guard relay bandwidth for *Single Guard*. The average of the median guard set bandwidths over the year 2015 are 34 MBps and 49 MBps for *AS* design and *BW* design, respectively. Although we used the same bandwidth thresholds τ_{up} and τ_{down} as Hayes and Danezis [14], *BW* design's guard sets tend to have more bandwidth because bandwidth is the primary consideration for managing guard sets. In *AS* design, network location is the primary consideration and bandwidth is secondary.



(a)

Figure 4.15. CDF of guard sets' bandwidths..



(a)

Figure 4.16. Bandwidths of guard sets over time..

Performance. To evaluate the performance of the guard selection mechanisms, we simulate them using Shadow [41, 42], a discrete-event network simulator that runs real applications like Tor and BitCoin on a single machine. Using Shadow, we simulate *Single Guard*, *AS* design, and *BW* design on a Tor network with 742 relays (including 152 guard relays) and 2700 clients (including 2280 web clients).

The median of the times to download the first byte (TTFB) for clients are 0.55, 0.55, and 0.56 seconds in *AS* design, *Single Guard*, and *BW* design, respectively. Thus, the responsiveness is about the same in all three approaches. The median times to download the last byte for web clients (TTLB) are 1.221, 1.185, and 1.183 seconds in *BW* design,

AS design, and *Single Guard*, respectively. Here, all the mechanisms have almost the same throughput.

4.6 Discussion

In this section, we discuss the implications of our findings and the scope for future work.

Deployment. To implement guard sets in Tor, the Tor directory authorities would be responsible for building and disseminating the guard sets. In particular, they would use information on p2c links from CAIDA [47] to identify customer cones and build the tree structure of *AS* design, including Root Sets, Branch Sets, and Guard Sets. They then ship the information about the design structure such as list of Root Sets, Branch Sets, Guard Sets, and which guard relays are in the Guard Sets to the clients using the consensus documents. The clients will get the information about the guard sets through the consensus documents. Then clients only need to pick their guard set using the algorithm mentioned in Section 4.4.5. We propose a format for adding guard set information to the documents. As we mentioned, the directory authorities can build the guard sets and ship them to the clients through the consensus documents. We propose the following format for adding the guard set's information to the consensus documents. This format is compatible with the formats used in consensus documents explained in [69].

Each router entry in the consensus documents contains a set of items. Each item sits in a separate line which is started with an identifier. To include the guard set information to the consensus documents, we add one more item for each guard relay. This item has the following format:

```
"g" SP ROOTSET-ID SP BRANCHSET-ID SP GUARDSET-ID NL
```

Where:

"g" = the identifier.

SP = white space.

ROOTSET-ID = 16–digit unique Root Set identity.

BRANCHSET-ID = 16–digit unique Branch Set identity.

GUARDSET-ID = 16–digit unique Guard Set identity.

NL = new line.

We used this format to add the guard set information to consensus documents in the year 2015. We observed that the average document size was increased 43 Kbytes (from 1.507 to 1.550 Mbytes).

Since not all Tor clients will upgrade at once, some degree of incremental deployment is needed. One approach is to have the directory servers provide just the existing consensus documents to older clients and the additional guard set information to upgraded clients. To prevent a major partitioning of clients into anonymity sets based on different behavior, Tor can wait until many clients have upgraded before setting a flag that initiates the use of guard sets. It may be best if transition to guard sets happens slowly, as users could continue with their current guards and not rotate from them prematurely. Beyond this, further studies should be conducted on the topic of incremental deployment to understand the impact of different transition strategies on the anonymity of users and performance of the system.

Orphan guard ASes. There are some cases in which guard ASes are isolated in a customer cone by themselves. In such cases, our algorithm may not be able to group these guard ASes with other guard ASes, forcing them to form smaller guard sets. If these sets do not offer enough bandwidth, they pose a risk for guard fingerprinting. To mitigate this issue, we can ignore these low bandwidth guard sets until enough guard relays join the

network such that the low bandwidth guard sets will be merged to form a sufficiently high bandwidth guard set. In our one-year simulations, we observed a median of two such ASes and a maximum of six ASes, and the median bandwidth of these ASes were 3.6 MBps.

AS relationships and customer cones. Customer cones are not as simple in practice as the tree model would suggest. Given the CAIDA AS relationship graph, which itself is not completely accurate, the cones must be inferred based on some assumptions. For our work, we apply the recursive customer algorithm [62, 61] (see §4.2.1). Other approaches might yield different results, but the key feature we need is that the attacker cannot add guards to arbitrary points in the hierarchy. Issues such as accuracy of the cones and the presence of p2p traffic passing through IXPs should not affect our overall findings much because they affect the AS graph but not the relative stability and hierarchical nature of the AS graph that is being exploited by the proposed technique.

We note that relying on a single organization such as CAIDA to form customer cones may be vulnerable to attacks on the organization or its information. Further thinking and experimentation should be performed before deploying these techniques in Tor.

To evaluate the risk of AS-level adversaries, we used Qiu and Gao AS-level path inference [43]. Although this type of inference can be inaccurate in identifying all of the ASes on a path [70], Barton and Wright report that it is 90% accurate in identifying the eight most common ASes that appear on both ends of a Tor path [59]. Further, they find that these eight ASes account for about 98% of all instances of an AS appearing on both ends of a Tor path. Thus, our findings both with and without DeNASA should provide a reasonably accurate estimate of the risk of network-level attacks.

Finally, we note that IP-to-AS mapping is not perfect. BGP is not secure, which can be leveraged for attacks on Tor [44]. Thus, it can be similarly attacked to undermine the *AS* design, and the importance of this requires further investigation.

Virtual Hosting. *AS* design takes advantage of this fact that the placement of guard relays in a given AS is harder than manipulating the bandwidth, which is the approach offered by *BW* design. There are some organizations and hosting providers like OVH that contribute a significant fraction of guard relays to the Tor network.

organization	BW	Relays	ASes	VPS
ONLINE S.A.S.	4511	141	1	✓
OVH SAS	4110	227	1	✓
Hetzner Online GmbH	2745	122	1	✓
Digital Ocean, Inc.	1367	63	7	✓
myLoc managed IT AG	813	28	1	✓
SURFnet	674	9	1	×
SG. GS	610	6	1	×
LeaseWeb Netherlands	489	18	2	×
domainfactory GmbH	438	16	1	✓
Init7 (Switzerland) Ltd	404	12	1	×
ISPpro Internet KG	336	17	1	✓
Contabo GmbH	240	17	1	✓
ITL Company	203	13	2	×
Strato AG	195	12	1	✓
SoftLayer Tech. Inc.	188	12	2	✓
PlusServer AG	186	12	1	✓
Cogent Com.	184	6	1	×
SUNET Swedish Uni	178	6	1	×
Free SAS	162	7	1	×
WorldStream	162	5	1	✓

Table 4.1. Top 20 organizations, sorted by bandwidth, running guard relays in Sept. 1, 2015. We used CAIDA dataset to map the ASes to the organizations [1]. The columns are the organization's name, the organization's bandwidth, the number of relays, the number of ASes on the organization, and whether the organization is a VPS or Not.

Table 4.1 shows the top 20 organizations running guard relays. The adversary can run a guard relay on hosting provider like OVH and get into the guard sets built based

on OVH's AS. This allows the adversary to target those guard sets beyond what we have discussed in this paper. On the other hand, for a large provider like OVH, there will be multiple guard sets, and no single one can be targeted. Further, the adversary does remain confined to his Branch Set and Root Set. Closer examination of this issue is needed before *AS* design can be deployed in Tor.

DoS Within Sets. In *AS* design, if the client's guard set is dismantled, the client will select another Guard Set from her Branch Set. This confines the adversary better to her Branch Set and limits the number of clients the attacker can compromise. On the other hand, the adversary can DoS other Guard Sets in the same Branch Set and force their clients to pick her Guard Set. This can be limited by picking the new guard set from a larger region in the network, such as anywhere in the Root Set. Since this creates more opportunities for the adversary to compromise guard sets, and since DoS attacks are active and detectable, we argue that it is better to select from the Branch Set instead.

Guard set rotation. Like Hayes and Danezis [14], we did not include guard set rotation in our system since clients may rotate to malicious guard sets [46]. On the other hand, without guard set rotation, old guard sets will collect more users over time compared to new guard sets. Moreover, we do not let compromised clients regain their privacy by rotating their guard sets. This trade-off also exists in other guard selection schemes. In Tor's single guard selection policy, the client should rotate her guard every 9 to 10 months. This guard rotation period is long enough that guard rotations are significantly reduced, yet compromised clients eventually regain their privacy. Such a guard rotation policy can also be applied to the guard set designs. We can also consider an age metric for guard sets that keeps the number of clients from growing too much in old guard sets.

CHAPTER 5

Defense against Website Fingerprinting attacks

5.1 Introduction

Tor is known to be vulnerable to traffic analysis attacks. The adversary who observes both the entry and exit sides of the traffic is able to correlate the traffic and link the client to her destination. This type of traffic analysis attacks needs powerful adversaries. A branch of traffic analysis attacks is *Website Fingerprinting* (WF) attacks. The goal of the adversary in WF attacks is to identify which websites the client is visiting.

The WF attacker is considered to be a *local* and *passive* attacker. The *local* means that the attacker is located in the client's network, and knows her IP, for example, she can be the client's wireless router or cable/DSL modem, the client's ISP, the guard node itself, and an AS between the client and guard node. The *passive* attacker means that the attacker only eavesdrops the traffic and does not manipulate the traffic, for example, by reshaping the traffic or tagging the traffic. Being passive makes the WF attacker almost impossible to detect. Because the WF attacker is assumed to be local and observes only the entry side of the traffic, the WF attacker is considered to be a weak adversary and makes it a serious threat against Tor.

The WF attack is a supervised classification problem. The websites are the labels and the traffic traces are the instances or observations. In this work, we assume that the client is browsing the web through Tor network. The attacker uses the same privacy enhancing technology as the client, the Tor network, to collect a set of instances for the websites that she is interested in. From the collected instances, the attacker extracts a set of pre-defined features and trains a classifier on the extracted features. Then the attacker observes

the client's traffic, extracts the features from the traffic, and classifies the traffic with the trained classifier.

The WF attacks have been improved over time from both feature extraction perspective and the classifiers power [15, 16, 17, 18, 19, 20, 71]. The accuracy rate of the state-of-the-art WF attack now reaches 98% [72].

In response to the threat of WF attacks, there have been proposed several defenses against WF attacks on Tor [73, 74, 75, 76, 77, 78]. The WF defenses try to change the pattern of the traffic in a way that confounds the classifier. The change in the pattern of the traffic can happen by the link padding (the packet padding is already implemented in Tor as Tor cells are padded to 512 bytes). In the link padding strategy, dummy packets are sent to change the pattern. The WF defenses have used different strategies in applying the link padding. In BuFlO family defenses (including BuFLO [79], CS-BuFLO [77], and Tamaraw [78]), the upload and download transmissions happen in the fixed rates. These defenses buffer the packets and send them at the fixed rates, whenever there is no scheduled packets in the buffer to be sent, a dummy packet is sent. This type of defense is known to be too expensive in terms of bandwidth usage and latency. The super-sequence family defenses [19, 76, 73, 80] cluster the traffic into few sets. All the traffic traces in the same set are padded to the minimum sequence that contains all the sequences on that set. Beside the bandwidth and latency overhead, the problem with these defenses is that they need a large database of webpage templates for building the sets. Maintaining this database and distributing it in the Tor network is very challenging.

WTF-PAD [81] is an effective defense against WF attacks with reasonable overheads, such that it would be practical for deployment in Tor. WTF-PAD injects the dummy packets to fill the gaps in the traffic and creates fake bursts. Because WTF-PAD does not delay the real packets, it does not add latency overhead, and only comes at a fair bandwidth overhead cost (around 65%). WTF-PAD successfully could drop the accuracy rate

of kNN [19] from 90% to 17%. A recent study [72] using a Convolutional Neural Network (CNN) could break WTF-PAD and achieve accuracy rate 90% on the traffic traces protected by WTF-PAD. In this work, we are going to introduce a new defense strategy using adversarial samples generated by a deep neural network. We generate the adversarial examples using the techniques in the computer vision field. We consider two different scenarios to evaluate the effectiveness of the adversarial examples as WF defense. In the first scenario the attacker is not aware of the defense and has been trained on the non-defended traces. In the second scenario we consider the case that the attacker has trained the classifier on the defended traces. We show that the adversarial examples generated for the traffic traces are not an effective defense in the second scenario. To solve this problem we propose a new method to modify the samples that causes misclassification in the classifier with small amount of bandwidth even if the attacker is trained on the defended traces. Our defense could drop the accuracy rate of state-of-the-art attack from 98% to 60% (22% worse than the state-of-the-art WF defense) with 47% bandwidth overhead (31% better than the state-of-the-art WF defense)

5.2 Related work

In this section we explain the previous researches in the WF. We categorize the WF researches into WF attacks and WF defenses.

5.2.1 WF attack

Website Fingerprinting has been studied extensively. The early researches in this field date back to 1990s when the researchers evaluated the information leak about the URL through the encrypted HTTP requests [82, 83]. The first attempt of WF attacks against Tor was performed by Herrmann et al. [84]. Their attack used Naive Bayes classifier with the frequency of the packet lengths as the feature set. They evaluated the attack in the

closed-world setting with 775 sites. Their attack was not successful and it reached only %3 accuracy rate. The reason for this low accuracy is that they only used the packet length as the feature. Tor already sends the traffic in the fixed 512-byte packets, known as *cell*, which makes this feature ineffective. Since then the researchers revisited the attack from both the classifier's and feature set's point of view. The WF attack and WF defenses are evaluated into two different settings: *closed-world* and *open-world* setting. In the closed-world, we assume that the client is visiting a small set of websites, called *sensitive* or *monitored* sites, and the attacker is trained the classifier on this set of websites. In the open-world scenario, we assume that the attacker is trained the classifier only on *monitored* sites, but the client is visiting any website including *monitored* or *non-monitored* sites. Over time the performance of the attacks has been improved such that they could reach over 98% accuracy rate in the closed-world scenario and their false positive rate is less than 0.4% in the open-world setting. For the rest of this section we have selected the state-of-the-art WF attacks that are being used to benchmark other WF attacks and defenses.

k-NN attack: k-NN attack was introduced by Wang et al. [19]. The attack uses a modified version of k-NN classifier. Instead of simply using Euclidean distance, the attack uses a weighted distance function to measure the proximity of the samples. The attack has two phases, weight learning phase, and classification phase. The weight learning process is an iterative algorithm that the weights are learned. Weights indicate the importance of each feature in computing the distance . In the classification phase, the learned weights are used to compute the distances in the k-NN. One of the most important contributions of this work is the feature set. They defined a diverse set of features which contains general information about the traffic, packet ordering, concentration of the packets, and bursts in the traffic. Using this large feature set with weight-adjusting k-NN, they obtained 91% accuracy in 100-site closed-world setting.

CUMUL attack: Panchenko et al. [85] proposed the CUMUL attack. The attack uses Support Vector Machines (SVM) as the classifier. SVM had been used in the previous work but Panchenko et al. [85] used the SVM with a new feature set. They used the sequence of cumulative sum of packet sizes in the traffic as the feature. They also collected a data set that is more representative of the real sites browsed in the Internet. In contrast to prior work that they simply use most popular websites from Alexa.com [49], Panchenko et al. assembled a list of websites from different sources such as trend links in Twitter, trends in Google, random pages from Google searches, and censored sites in China. Moreover, for the first time they differentiated the website fingerprinting from webpage fingerprinting. They evaluated the detectability of both single webpages and complete websites. Their attack obtained 92% accuracy rate in the closed world setting. To evaluate the effectiveness of our WF defenses, we selected the CUMUL attack as one of the WF attacks to test the defenses against with.

k-Fingerprinting (k-FP): k-FP was proposed by Hayes and Danezis [86]. They put all the previously defined features in the literature together and added some other features, such as the statistics on the timestamps and the volume of the traffic, to their feature list. They used Random Forest to rank the features and found the most important features in traffic. Then they used the important features to train their classifier. The classifier first learns the fingerprints using Random Forest, the fingerprints are the leafs of the trees in Random Forest. Later the fingerprints are used by k-NN to do the classification task. Their attack could reach 91% accuracy in the closed-world setting.

All the WF attacks that we described so far used the traditional machine learning algorithms. Since 2016 the researchers have brought Deep Learning (DL) to the WF field. There are few researches that they have applied deep neural networks in the WF. In the next few paragraphs, we will explain these attacks.

SDAE attack: Deep learning was used for the first time by Abe and Goto [87]. The studied the application of Stacked Denoising Autoencoders (SDAE) in WF attacks. Their attack's accuracy rate was 88% and it was lower than the previous work. The reason of their low accuracy rate is that they used a small dataset collected in [19] to train the SDAE and deep neural networks need more data to train compared to other classification algorithms.

Rimmer et al. attack: Rimmer et al. [88] proposed using the deep learning to bypass the feature engineering phase of traditional WF attacks. In order to use the DL, they collected a large dataset of 900 sites and 2,500 traffic traces each. They applied different DL algorithms, such as SDAE, Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM), on the traffic traces. They found that CNN outperforms the other DL algorithms and obtains 96% accuracy rate in the closed-world setting.

Deep Fingerprinting (DF): Sirinam et al. [72] extensively evaluated the use of DL in the WF. They developed a deep CNN model that it could outperform all the previous models and reach up to 98% accuracy rate in the closed-world setting of 100 sites with 1,000 instances each. They also evaluated their model against the state-of-the-art WF defenses, and they showed that DF can still outperform the previous attacks even when a defense is in place. Their attack could obtain 90% accuracy against WTF-PAD [81], a potential WF defense candidate to be deployed on Tor.

5.2.2 WF defense

To defeat the WF attackers, the WF defenses generate a cover traffic to hide the features in the traffic. The defense mechanisms generate the cover traffic by sending dummy packets or delaying the real packets. Sending the dummy packets comes at the cost of bandwidth overhead in the network and delaying the real packets causes the latency overhead in the download time and hurts the users experience. Therefore, there have been several

studies that tried to balance the trade-off between the WF defense's overhead and efficacy of the defense against WF attacks. In this section we explain some of the state-of-the-art WF defenses.

BuFLO family defense: This family of defenses transmits the traffic in the constant rate in both directions, download and upload traffic. Their main differences are in the stop condition and rate adjustment. This family includes BuFLO [20], CS-BuFlO [77], and Tamaraw [78]. BuFLO is the first defense of this kind and it sends the packets in the same constant rate in both directions. It stops the transmission when the page is loaded and a minimum amount of time has passed. The overhead of the traffic is governed by both the transmission rate and the minimum time threshold in the stop condition. Moreover, although the defense covers the fine-grained features like burst information, the course-grained features like the volume and load time of the page still leak information about the website. To reduce the amount of overhead in BuFLO, Tamaraw and CS-BuFLO proposed to transmit the download and upload packets in different fixed rates. To provide better cover traffic, after the page is loaded, Tamaraw keeps padding till the total number of transmitted bytes is a multiple of a certain parameter, and CS-BuFLO pads the traffic up to a power of two, or to a multiple of the power of the amount of transmitted bytes. BuFLO family defenses are expensive in terms of overhead, and they add 2 or 3 times latency overhead and more 100% bandwidth overhead.

Super-sequence family defense: This family of defense finds the super-sequence of the traffic traces. These defenses define some anonymity sets and cluster the websites into these anonymity sets. For each cluster they find a representative sequence for that cluster such that it contains all the sequences in the cluster. All the websites that are falling into the same cluster are molded to the representative sequence. This family includes Supersequence [19], Glove [89], and Walkie-Talkie [90]. Super-sequence and Glove use approxi-

mation algorithms to find the super-sequence and they apply the molding directly to the cell sequences that causes higher bandwidth and latency cost. Walkie-Talkie, uses anonymity sets with size two. It applies the molding in the burst sequences instead of the cell sequences. Walkie-Talkie (WT) requires the sequences of the bursts in the traces. Therefore, the communications between the client and server should be in the half-duplex mode. WT enforces the browser to half-duplex communication and only allows the browser to send new requests if the responses of all the previous requests have been received. WT molds the burst sequence of a sensitive website to a burst sequence of non-sensitive website and vice versa, which theoretically the accuracy rate of the WF attacks never goes beyond 50%. To mold a burst sequence to another one, WT picks the maximum burst length from both bursts sequences in the current burst location and sends a burst with that size. WT reports 31% bandwidth overhead and 34% latency overhead, which this latency overhead comes from the half-duplex communication and drops the CUMUL's accuracy rate from 64% to 20%.

Adaptive Padding (AP): Shmatikov and Wang [91] proposed Adaptive Padding (AP) as a countermeasure against end-to-end traffic analysis. Juarez et al. [81] extended the idea of AP and proposed WTF-PAD defense as the adapted version of the AP to protect the Tor traffic against the WF attacks. WTF-PAD tries to fill the large gaps in the inter-arrival packet times. Whenever there is a large inter-arrival packet time, WTF-PAD sends the dummy packets. WTF-PAD previously computed the histogram of inter-arrival packet times. If the inter-arrival packet time is larger than a random inter-arrival time drawn from the histogram, the WTF-PAD sends a dummy packet to fill the gap. This limits the amount of bandwidth overhead required and does not incur any latency costs. WTF-PAD also sends dummy bursts, the consecutive dummy packets, in the large gaps. They showed that WTF-PAD can drop the accuracy rate of k-NN attack from 92% to 17% with a cost of 60%

bandwidth overhead. Sirinam et al. [72] showed that their attack, DF, can achieve up to 90% over WTF-PAD.

Application Level defenses: Cherubin et al. [92] proposed the first WF defense designed in the application layer. They proposed two WF defenses in both client-side and server-side. The server-side defense does not need any action from the client and it needs to be deployed in the servers. Because Onion sites are more concern about the privacy of their clients, Cherubin et al. argued that their defense is more suitable for Onion sites. *Application Layer Padding Concerns Adversaries (ALPaCA)* is the server-side defense that alters the size distribution for each content type, e.g. PNG, HTML, CSS to an *average* Onion site. The size distributions are obtained from all of the Onion sites. In the best case, this defense leads to 41% latency overhead and 44% bandwidth overhead which drops the CUMUL's accuracy rate from 56% to 33%. The server-side defense may have slow deployment over Onion sites. Thus, the authors proposed a client-side defense, *Lightweight application-Layer Masquerading Add-on (LLaMA)*. LLaMA adds random delay in the HTTP requests. These delayed requests change the order of the packets, requests and responses in the traffic. LLaMA drops the accuracy of the CUMUL attack from 56% to 34% at cost of 9% latency overhead and 7% bandwidth overhead.

5.3 Background

Deep neural networks (DNNs) have shown that they can outperform the other machine learning algorithms in many tasks [93, 94, 95, 96, 97, 98, 99]. The high performance of the DNNs mostly comes from the advantage of large datasets available and hardware accelerations. DNNs need less feature engineering and expert knowledge, and can extract the intricacy of the data from the raw data. DNNs have also shown significant progresses

in the WF attacks fields [72, 71] and they are independent of the hand-crafted features used in the previous WF attacks [86, 85, 19].

A neural network can be modeled as a function $F(x) = y$ which takes $x \in R^n$ as the input and returns $y \in R^m$. F is the model and depends on the set of model parameters θ. We use the neural network as *m*–class classifier.

The output of the neural network is computed using a softmax function. The output of the neural network is a vector of $(y_0, y_1, y_2, , y_{m-1})$ which $0 \leq y_i \leq 1$ and $\sum_{i=0}^{m-1} y_i = 1$. Vector *y* shows the probability distribution of the predicted label of sample *x* over all the classes. The classifier assigns label $C(x) = argmax_i F(x)_i$, which has the highest probability over all classes, to sample *x* . We depict the true label of sample *x* as $C^*(x)$. We define the output of a layer in the neural networks, except the softmax function, as $Z(x) = z$, then the neural network including the softmax function is as follows:

$$F(x) = softmax(Z(x)) = y$$

The input to the softmax function is called *logit*. A neural network consists of multiple layers can be formulated as follows:

$$F = softmax \circ F_n \circ F_{n-1} \circ ... \circ F_0$$

Where:

$$F_i(x) = \sigma(\theta_i \cdot x) + \hat{\theta}_i$$

σ is the activation function which can be tanh [100], sigmoid, ReLU [101], or ELU [102]. θ and $\hat{\theta}$ are the model weights and the model biases, respectively.

We are going to use the vulnerability of machine learning algorithms to *adversarial examples* [103, 104, 105] as a defense against the WF attacks. Adversarial examples are crafted inputs that cause misclassification in the machine learning algorithms . The adversarial examples have the *transferability property* that the adversarial examples crafted on a specific machine learning model are highly effective against the other machine learning models [106]. This property leads to black-box attacks against the machine learning

models that the attacker has no idea about the architecture or the parameters of the victim model, and has only access to the output of chosen inputs. Papernot et al. [106] developed the generalized black-box attacks against a wide class of machine learning models and evaluated within and between classes transferability of the adversarial examples. They showed that the adversarial examples are highly transferable. Moreover, most of the techniques in generating adversarial examples leverage the gradient of the training algorithm, which is used to update the weights, to modify the input and cause the misclassification in DNNs [104, 105, 107]. Therefore, in order to create the adversarial examples over traffic traces, we consider a DNN model as a WF attacker and generate adversarial traces based on this model. We evaluate the effectiveness and transferability of the crafted adversarial traces against the other WF attacks.

There are few DNN models that are introduced as WF attacks [72, 71]. Sirinam et al.'s model [72] is the state-of-the-art model and could outperform all the previously designed WF attacks. Their model is a deep Convolutional Neural Network (CNN) with 8 convolutional layers and 3 full connected layers. Because their model is deep, using this model to craft the adversarial traces would slow down the process. Therefore, we define a CNN model that is not too deep but provides us descent accuracy rate. We use our defined model to generate adversarial traces and then we test the transferability of the generated traces to other models, such as CUMUL attack (an SVM model) and DF attack (a deep CNN model).

5.3.1 Convolutional Neural Networks (CNNs)

CNNs are similar to the regular neural networks and consist of multiple stacked layers of neurons with trainable weights and biases. In the regular neural networks, each neuron is connected to all the neurons in the previous layer. The neuron performs a dot product between the weights and the previous layer, and passes the output from a non-

linear function, called the activation function. The whole neural network is like a function that gets an input and returns the class scores.

Because each neuron is fully connected to the previous layer, the regular networks are not scalable very well. In the first layer of the regular neural networks, as the input size increases, the number of weights increases because the weights in the first layer corresponds to the number input features. This fully connected manner leads to large number of trainable weights, complexity of the model, and overfitting. In contrast, in CNN, each neuron is connected to small region of the previous layer that causes the number of trainable weights to be independent of the input size.

There are three main layers to build the CNN architecture, *convolutional layers*, *pooling layers*, and *fully connected layers*.

Convolutional layers consist of a set of filters. Filters in each layer have fixed lengths and they get connected to small region of the previous layer. Each filter is convolved with the input. The convolution is the dot product of the weights (and the bias) of the filter with a small region of the previous layer overlapping with the filter. The filter drifts slowly over the input and the dot product is computed again. The output of convolution of the filters are passed through an activation function to add non-linearity to the network. Standard activation functions are Sigmoid and Rectified Linear Unit (ReLU), tanh, Leaky ReLU (LReLU), Parametrized ReLU (PReLU) and Exponential Linear Unit (ELU). The output of the convolutional layer followed by the activation function is called *feature maps*. Each filter in the convolutional layer learns different features that stacking them together provides the different representations of the input.

The feature map further passes through a *pooling layer*. The pooling layer performs a down-sampling operation along the spatial dimensions, which leads to the reduction in the computational cost. The pooling layer combines several values into single value by getting either their maximum (*max pooling*), or their average (*average pooling*). Because

this layer smashes several values of features into the same bin, they make the model more robust to shifts in the input and generalizes the model.

Several convolutional layers followed by the activation function and pooling layer are stacked together. Each of these combined layers (convolutional layer + activation function + pooling layer) learns different representation of the data. In the very first combined convolutional layers, they learn the low level representation of the data, and as we go deeper these layers learn high level representation of the data. The number of trainable variables (weights and biases) depends on the size of the filters and the number of filters in the convolutional layers. Therefore, the number of the trainable variables are much smaller than regular neural networks that enables us to go as deep as 10,000 layers [108].

The output of the combined convolutional layers is then attached to a set of *fully-connected layers*. These fully-connected layers are like the regular neural networks and their main task is to compute the classification scores.

Although CNNs have small number of trainable variables compared to the regular neural networks which make them faster to train or more generalized, other techniques used in the regular neural networks for improving the performance and regularization, such as *Batch Normalization* and *Dropout*, are widely used in CNNs. Batch Normalization normalizes the output of hidden layers that causes better optimization and makes the model more robust. The Dropout technique randomly shuts down a fraction of neurons in the hidden layers in the training phase. The Dropout actually makes the training of the model harder by adding randomness to the training and does not let the model memorize the data.

To generate the adversarial examples, we define a simple and shallow CNN model. We want our model to be simple and fast. Our architecture contains three convolutional layers followed by two fully-connected layers. We used ReLU as the activation function on our architecture. Table 5.1 shows our model architecture. This model gives us 92% accuracy rate on our dataset, we will discuss about our dataset in Section 5.3.2.

Table 5.1. MODEL ARCHITECTURE

Layer type	size
Convolution + ReLU	$1 \times 8 \times 32$
Convolution + ReLU	$1 \times 8 \times 64$
Convolution + ReLU	$1 \times 8 \times 128$
Fully Connected + ReLU	512
Fully Connected + Softmax	83 (number of classes)

5.3.2 Dataset

We apply our algorithms for generating adversarial examples on the traffic traces on the burst level. The bursts are the sum of consecutive packets in the same direction. To get the traffic traces in the burst level, we can't simply convert traffic traces, which are the sequences of packets, to the sequence of bursts by adding up the consecutive packets in the same direction. The traffic traces in [19, 92, 86] were collected as the communications between clients and servers were in the full-duplex mode. This means that the incoming (outgoing) bursts might be interrupted by outgoing (incoming) bursts because before receiving the full response from the previous request, the client may send new requests. We need to collect the traffic on the half-duplex communication. Walkie-Talkie (WT) [90] also works on the half-duplex communication and it finds the supersequence in the burst level. Sirinam el al. [72] collected a big dataset of traffic traces over the half-duplex mode. Their dataset contains 100 sites, top 100 sites in Alexa.com [49], with 900 instances for each class. This data collected in half-duplex mode over Tor network. For our evaluation, we use their dataset. We cleaned their data and removed the traces shorter than 50 packets, and the ones that their first packets are incoming packets. After the cleaning process we ended up with 83 classes with 720 instances per class. Moreover. Sirinam el al.'s dataset contains 40,000 instances from 40,000 different sites, one instance per site. We call this set of traces as Open-World dataset.

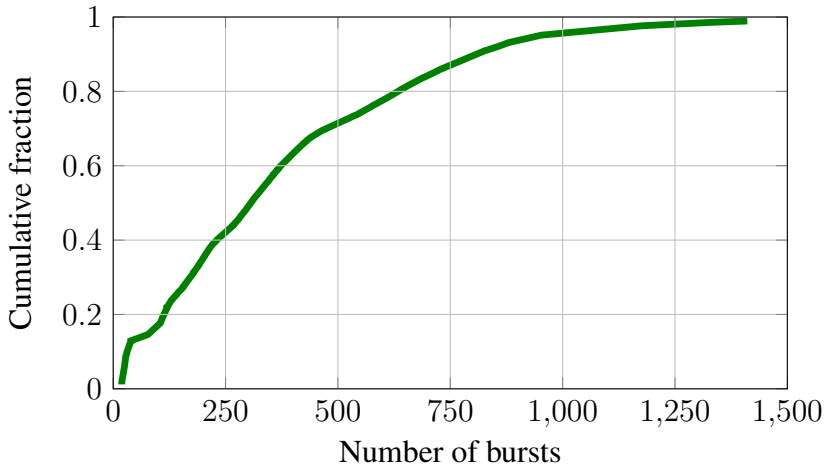


Figure 5.1. The CDF of the number of bursts in the traces.

The length of the burst sequences are different through the different websites or even through different visits of the same website. The input to the machine learning models needs to be in the fixed sizes. We need to consider a fixed burst sequence length for all the traces. Figure 5.1 shows the CDF of the burst sequence lengths in our dataset. The figure shows that more than 80% of traces have less than 750 bursts. We examined the accuracy rate of the model, described in Section 5.3.1, with the input size of 750 bursts and 1500 bursts, in both cases the accuracy rates were 92% and 93%, respectively. The difference between the full size input (1500 bursts) and its short form (750 bursts) is negligible. To decrease the computational cost and speed up the process, we used the input size of 750 bursts in our evaluations.

Note that the defense that we are proposing is more suited for Onion sites because Onion Sites are more willing to collaborate in deploying the defense on their servers to protect their users. For the sake of evaluation we used the data from the regular web. We believe that the results should be consistent on the Onion sites given that the WF attacks are less effective on Onion sites [109], which means the defenses are more effective.

5.3.3 Threat Model

We assume that the client browses the Internet through the Tor network to hide her activities (see Figure 5.2). The adversary that we are investigating is *local* which means the attacker already knows the identity of the client and his goal is to detect websites she is visiting. A *local* adversary can be an eavesdroppers on the users local network, local system administrators, Internet Service Providers (ISP), Autonomous Systems (AS) between the user and the entry node, and the operators of the entry node. Moreover, we suppose that the adversary is *passive*, meaning that the adversary only taps the traffic between the client and the entry node, he just observes and records the traffic traces, he does not drop, delay, or modify the real packets, and he does not inject fingerprints in the traffic traces.

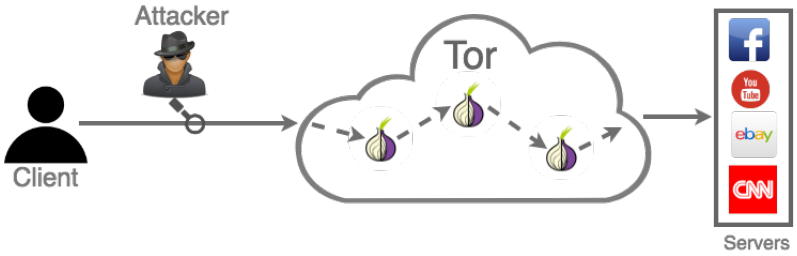


Figure 5.2. WF attack.

WF defenses require two parties in the network to control the both sides of the traffic, upload traffic and download traffic. We assume the part of the defense controlling the upload traffic is located in the client side. The part of the defense controlling the download side of the traffic can be deployed on a *bridge*. The *bridge* is located somewhere between the adversary and the webservers. The possible options for deploying the *bridge* can be the entry node or webservers. We assume that the padding packets by the WF defenses are not distinguishable by WF attackers from the real packets and they are performed by two trusted end points of the WF defense.

5.4 Adversarial Examples as WF defense

Szegedy et al. [110] first discovered that several machining models, including neural networks, are vulnerable to the *adversarial examples*. These machine learning models misclassify the adversarial examples with high confidence. The adversarial examples are the crafted inputs with slight, but intentional, perturbation to the correctly classified samples from the data distribution. In many cases, adversarial examples can be general and *transformable*, the adversarial examples crafted for a certain machine learning model are misclassified in other models trained on different subsets of the data, even they are sometimes misclassified to the same class.

The goal of the adversarial examples is to generate samples that are similar to the samples of a certain class but misclassified to other classes. Given an input sample *x* and target class *t* ($C^*(x) \neq t$), the goal is to find $x^{'}$ which is close to *x* according to some distance metrics and $C(x^{'}) = t$. In this case, $x^{'}$ is called the *targeted* adversarial example because $x^{'}$ is misclassified to a certain target label *t*. The other class of adversarial examples are *untargeted* adversarial examples that we are interested to generate example $x^{'}$ which is mis-classified to any other classes except the true class ($C^*(x)$).

The reason behind the existence of adversarial examples is a mystery. There are several speculations about the cause of the adversarial examples, such as extreme non-linearity of deep neural networks, overfitting, insufficient model averaging, and insufficient regularization. There is an arm race between defenses against adversarial examples and the new types of adversarial examples. Carlini et al [24] showed that none of the recent defenses against adversarial examples work and they can be broken.

Based on the fact that the adversarial examples are transformable and strong such that can avoid the defenses, these examples can be a good candidate to act as a defense against WF attacks. The goal in WF defenses is to insert dummy packets to the traffic to fool the classifier, the WF attacker. The number of dummy packets in the traffic should be as small

as possible to keep the bandwidth overhead low. These properties of WF defenses, low overhead and misclassification, correspond to the properties of the adversarial examples, which are generated based on a small perturbation to the examples to fool the classifier.

Challenges in Adversarial Examples In order for designing a WF defense, the adversarial examples can be generated from web traffic traces. Generating adversarial examples from web traffic traces is more challenging than adversarial examples in other fields such as image classification, voice recognition, and text classification. These challenges make the use of existing techniques in generating adversarial examples ineffective. The main challenges we face in crafting adversarial examples for web traffic traces are as follows:

- *Crafting adversarial examples on the fly*: In crafting the adversarial examples from traffic traces we are dealing with transmission of packets. The traffic trace is captured packet by packet and the adversarial examples should be generated as we send or receive the packets. The entire trace is not accessible to the algorithm to generate the adversarial example. The adversarial example should be generated on the fly. To tackle this challenge, we propose to generate adversarial traces in the burst level rather than the cell level. In this case, the server will provide the adversarial trace for its traffic and sends it back to the client. Both the client and server will follow the agreed traffic pattern in their communication.
- *No packet dropping, only insertion*: the perturbation to the traffic traces should be done only through the sending of the dummy packets. Dropping the real packets to perturb the traffic is not recommended due to the retransmission, bandwidth and latency overhead, and hurting the users experience.
- *Two parties are involved*: Traffic traces are a combination of download and upload streams. Both the client and the server do not have a full control on both streams. The download stream is mainly controlled by the server and the upload traffic is controlled

by the client. Therefore, the WF defenses have two elements to control both sides of the traffic, the client side and a bridge in the network. The client side controls the up streams and the bridge controls the down streams. Generating the adversarial traces in the burst level enables us to generate them in the server and send them to the client.

Addressing the above challenges may be too hard and inefficient without involving the web servers in generating the adversarial traces. Asking the regular web servers to take some of tasks in generating adversarial traces or traffic perturbation is infeasible because of inefficiency caused in their services for their non-Tor users. Instead of focusing on the generating adversarial traces for the regular web traffic, we propose to use adversarial traces for the Onion Sites traffics. Fortunately, the onion servers are more willing to contribute in defending their clients. Therefore, in following sections we propose methods to generate the adversarial examples for OSs.

5.5 Onion Sites

Onion sites, formerly known as *hidden services* are types of services provided by Tor that not only provide anonymity for the clients but also keep the servers anonymous. The location of onion servers are unknown by Tor. Onion sites provide various kinds of services such as web publishing, messaging, and chat.

Recent studies [109, 111] showed that the onion sites' traffic can be easily recognizable from the rest of Tor traffic. Therefore, the adversary can easily filter out the onion sites' traffic from the rest of Tor traffic and apply WF attack to detect the onion sites visited by the clients. Given that the number of onion sites is very small compared to the regular web, the WF attacker is dealing with a small open world. Thus, the WF attack on the onion sites are more serious issue compared to the rest of Tor.

Fortunately, the onion sites are more concerned than regular websites on the anonymity of their users, and they may be willing to contribute in defending their clients against potential threats like WF attack.

Cherubin et al. [92] leveraged this technique and developed a defense that is implemented on the onion servers. Their defense was effective at reducing the accuracy of state-of-the-art attacks from 70% down to 10-40% but it leads to 40-60% latency overhead. We propose to extend this approach by creating adversarial examples based on the server's knowledge of its own page profile. Creating adversarial examples for onion sites are less challenging. The onion server can generate the adversarial examples for its traffic and ship them to the client. Both, the client and server, follow that traffic pattern during the transmission.

5.5.1 Designing adversarial examples

To tackle these challenges, we first consider the traffic as a sequence of incoming (server to client) and outgoing (client to server) bursts. The burst themselves are a sequence of consecutive packets in the same direction. We can use the half duplex communication method proposed in Walkie-Talkie [90] to convert the traffic from a sequence of incoming and outgoing packets to a sequence of bursts. By this definition of the burst, we can increase the length of the burst by sending the dummy packets in the direction of the burst. Moreover, we can decrease the length of the burst by sending dummy packets in the opposite direction of the burst but in our methods we are not going to decrease the size of the real bursts. To generate adversarial traces over the burst sequences we borrow the techniques used in the computer vision for generating adversarial examples. Different algorithms have been proposed for generating adversarial examples in the computer vision, such as the Fast Gradient Sign Method (FGSM) [21], and Jacobian-Based Saliency Map Attack (JSMA) [105], and optimization-based methods [24, 112]. Carlini and Wagner pro-

posed [24] an powerful optimization-based algorithm to generate adversarial examples that can defeat the state-of-the-art defenses [113] against the adversarial examples. We used their technique and modified it based our needs to generate adversarial traces out of the burst sequences.

5.5.2 Adversarial Traces

Carlini and Wagner [24] proposed a strong method for creating adversarial examples in both targeted and non-targeted scenarios. Their method could bypass the state-of-the-art defense, the defensive distillation [113], against adversarial examples. Their attack algorithm is successful with 100% probability in both defended and undefended models.

Given a sample x and a model F, the algorithm finds a perturbation δ that makes $x' = x + \delta$ to be misclassified to any other class than $C(x)$ ($C(x) = argmax_i F(x)_i$), or in the case of targeted attack it is classified to target class t. The algorithm tries to find x' that is similar to x based on some distance metric D. The distance metrics can be L_0, L_2, or L_∞. The algorithm is formulated as below:

$$\begin{aligned} \text{minimize} \quad & \| \delta \|_p + c.f(x+\delta) \\ \text{such that} \quad & x + \delta \in [0,1]^n \end{aligned}$$

The algorithm will find δ such that minimizes the distance metric, which here is l_p norm, and the objective function $f(x+\delta)$. c is the chosen constant parameter to scale both the distance metric and the objective function in the same range. Carlini and Wagner [24] used binary search to find the proper c. Carlini and Wagner [24] defined several objective

functions, and at the end they found that the following objective function generates the best performing loss function: For targeted attack scenario with target class t:

$$f(x^{'}) = \max_{i \neq t}(F(x^{'})_i) - F(x^{'})_t$$

For non-targeted attack scenario as the true class for sample x is y, the objective function is:

$$f(x^{'}) = F(x')_y - \max_{i \neq y}(F(x^{'})_i)$$

Carlini and Wagner [24] showed that their algorithm can bypass the state-of-the-art defense against the adversarial examples. However, Carlini and Wagner in [107] showed that previously designed defenses can be defeated by constructing new loss functions, and the adversarial examples are harder to detect than previous thoughts.

5.6 Attack Scenarios

In order to evaluate the efficacy of the adversarial traces generated by the above-mentioned method, we consider two different attack scenarios based on when the defenses are applied. The defense can be applied either after the attack or before the attack. we evaluate the both scenarios in the following sections.

5.6.1 SCENARIO I: Defense After Attack

In this scenario, we assume that the attacker has trained its classifier on the traces that have not been defended. We assume that the attacker is unaware of a defense in place. Such a scenario can be valid in the case that we have an attacker that does not target any specific client and his goal is to identify the behavior of large number of users, but some of the clients may use some defense to protect their traffic, which the attacker is unaware of the defense or his goal is to monitor the majority of the clients.

Table 5.2. The evaluation of the defenses against the state-of-the-art WF attacks as the attackers are trained on the undefended traces

	BW overhead	Simple CNN	DF [72]	CUMUL [85]
Undefended	-	92%	98%	92%
Adversarial Traces	62%	2%	3%	31%
Walkie-Talkie(WT) [90]	69%	13%	23%	18.5%

For this scenario, we first train a classifier on our undefended traces and then we generate the adversarial traces. We examine the efficacy of the generated samples by testing them against the trained model.

In our evaluation we break the data into two sets, *Attacker Set*, and *Defender Set*, each set has 83 classes with 360 instances each. The attacker trains the classifier on the Attacker Set. The traces in Attacker Set are not protected by any defenses. The WF attacks that we apply on the Attacker Set are Simple CNN, DF, and CUMUL attacks. We chose DF attack as a state-of-the-art WF attack representing Deep Learning WF attacks, and CUMUL attack as the representative of the traditional machine learning algorithms, CUMUL attack uses SVM classifier and has high performance compared to other traditional WF attacks [72].

We apply the method described in 5.5.2 to generate adversarial traces from the traces in Defender Set, we call these traces in our evaluation as Adversarial Traces. We also apply Walkie-Talkie on Defender Set's traces. To generate Adversarial Traces, we use our Simple CNN as the target model (F) and the adversarial traces will be generated based on Simple CNN. We also apply Walkie-Talkie on Defender Set and evaluate the protected traces with Simple CNN, DF, and CUMUL attacks trained on traces in Attacker Set. Table 5.6.1 shows the results of our evaluations. According to the table, Adversarial Traces add 62% bandwidth overhead which is 10% less than Walkie-Talkie's bandwidth overhead (69%). Adversarial Traces generated for Simple CNN can confound the target model 98% of the times.

Table 5.3. The evaluation of the defenses against the state-of-the-art WF attacks as the attackers are trained on the defended traces

	BW Overhead	Simple CNN	DF [72]	CUMUL [85]
Undefended	-	92%	98%	92%
Adversarial Traces	62%	91%	97%	91%
Walkie-Talkie(WT) [90]	69%	38%	48%	39%

Moreover, the accuracy rate of DF and CUMUL attacks are 3% and 31%. This means that Adversarial Traces generated based on a target model with Simple CNN architecture can be highly transferable to other machine learning models. Almost all the adversarial traces generated by Simple CNN can confound DF attack, which is also a DL model. The results show that the adversarial traces are more transferable to DNN model than traditional machine learning models. The accuracy rate of DF and CUMUL attacks over traces protected by Walkie-Talkie are 23% and 18.5%, respectively. Adversarial Traces are more indistinguishable than Walkie-Talkie traces against the DF attack. However, Walkie-Talkie traces perform better against the CUMUL attack than Adversarial Traces.

5.6.2 SCENARIO II: Attack After Defense

In this scenario, we assume that the attacker knows that the client is using some sort of defense mechanisms to protect her traffic. The attacker then collects the traces protected by the same method as the client and trains his classifier with those traces. In this scenario, the training set and testing set are both traces protected by the same WF defense method. This scenario is more realistic because it has been shown that the effectiveness of the WF attacks defends on the attacker's knowledge of the clients [114]. Moreover, once a defense is deployed it is supposed to be accessible for all the users and used by all the users. Therefore, the attacker can also use the same defense as other clients. For evaluation in

this scenario, we protect the traces in Defender Set by Adversarial Traces (described in Section 5.5.2 using a target model with the architecture in Table 5.1) and Walkie-Talkie. Then we train the WF attacks, Simple CNN, DF, and CUMUL attacks, on 90% defended traces in Defender Set and test them with the remaining 10% of defended traces.

To generate Adversarial Traces, we train a target model with the same architecture as Simple CNN with the traces in Attacker Set. Thus, the target model (F) is trained on Attacker Set and used in generating Adversarial Traces on Defender Set. The generated traces will fool the target model. The results of the evaluation in this scenario are shown in Table 5.6.2. As shown in the table, even if Adversarial Traces are generated based on a target model with similar architecture as Simple CNN, they are highly detectable on Simple CNN as we train Simple CNN on the adversarial traces, and its accuracy is 91%. Moreover, DF and CUMUL attacks can also detect the adversarial traces with high accuracy rate, 97% and 91%, respectively. This means that generated adversarial traces are ineffective as the attacker is trained on the adversarial traces. Generating the adversarial traces works like a data augmentation technique in this case, if the attacker is trained on them, the attacker will detect them correctly. On the other hand, the traces protected by Walkie-Talkie are still highly undetectable in this scenario as the accuracy rate of DF and CUMUL attacks are 48% and 30%, receptively. However, training the attacker on the Walkie-Talkie traces improves the accuracy rate of the attack compared to Scenario I.

5.7 A New WF Defense Model

As we evaluated in the previous sections, generating adversarial traces using the proposed method in 5.5.2 is not effective as the WF attacker trains the classifier over the defended traces. In fact, generating adversarial examples works like the data augmentation

that if the attacker trains the classifier on the adversarial examples, he will be able to detect them.

In this section we introduce a new mechanism to perturb the traffic traces such that the classifier is not able to detect them. We borrow the same idea as the *targeted* adversarial examples. In order to defend a traffic trace, our defense picks a target sample and we keep changing the source sample (the trace we are going to defend) in a direction to get closer to the target sample and not being classified to the original class.

Assume that we have a set of sensitive sites $\mathcal{S}$ that we want to protect their traffic and we have a model $f(x)$ (called *detector*) that is trained on a set of data from $\mathcal{S}$ (we will later discuss the cases whether $f(x)$ should be trained on only sensitive sites or both sensitive and non-sensitive sites). We consider traffic trace I_s as an instance of source class $s \in \mathcal{S}$ that we want to alter it such that it is classified to $T = f(I_s)$ and $T \neq s$. I_s is a sequence of the bursts, $I_s = \left[b_0^I, b_1^I, ..., b_n^I\right]$. The only allowed operation on a burst, b_i^I, is to add some positive values, $\delta_i >= 0$, to that burst, $b_i^I = b_i^I + \delta_i$. The reason for $\delta_i >= 0$ is that we want to increase the volume of the bursts with sending dummy packets. If $\delta_i < 0$, it means that we should drop some packets to reduce the burst volume. Dropping real packets is not recommended due to intensifying the bandwidth overhead, delay, and retransmissions in the network.

In order to protect source sample I_s, we pick p random samples from other classes, $P_{I_s} = [{I^0}_{T_0}, {I^1}_{T_1}, ..., {I^p}_{T_m},]$. P_{I_s} is the pool for source sample I_s and it is a list of p randomly selected samples. ${I^j}_{T_i}$ is the j-th sample in the pool which belongs to target class $T_i \neq s$. We want to pick a target class and re-cast the source sample to be classified as that target class. To decrease amount of change to the source sample, which can be perceived as bandwidth overhead, we pick the nearest sample from the pool. Then we modify the source sample to move toward the nearest target sample.

We compute the l_2 norm distance between I_s and all the elements in P_{I_s}. From all the computed distances, we find the sample that has the minimum distance to the source sample, and we set it as our target sample I_T

$$D(x, y) = l_2(x - y)$$

$$I_T = \underset{I_t \in P_{I_s}}{\operatorname{argmin}} D(I_s, I_t)$$

Our goal is to increase the volumes of the bursts in the source sample such that the source sample is not classified as class s and the amount of change is as least as possible to lower the bandwidth overhead. To make the source sample to leave the source class, we move toward the nearest sample (I_T). We define Δ as the perturbation vector that we will add to the source sample to generate its defended form I_s^{new}.

$$\Delta = [\delta_0, \delta_1, \cdots, \delta_n] \quad (\forall i \in [0, \cdots, n] : \delta_i \geqslant 0)$$

$$I_s^{new} = I_s + \Delta$$

To find Δ that minimizes the overhead, we should minimize $D(I_s^{new}, I_T)$. To minimize the distance, we compute the gradient of the distance with respect to the input. The gradient points in the direction of steepest ascent, which makes the distance to be maximized. Therefore, we compute the gradient of the negative of the distance with respect to the input. The output of the gradient will give us the direction that we should move in the space to get to the target sample.

$$\nabla(-D(I, I_T)) = -\frac{\partial D(I, I_T)}{\partial I} = \left[-\frac{\partial D(I, I_T)}{\partial b_i}\right]_{i \in 0, \cdots, n}$$

Where b_i is the i-th burst in input I. To modify the source sample, we change bursts that their corresponding values in $(-D(I, I_T))$ is positive. Our perturbation vector Δ is:

$$\Delta = \begin{cases} -\alpha \times \frac{\partial D(I, I_T)}{\partial b_i} & -\frac{\partial D(I, I_T)}{\partial b_i} > 0 \\ 0 & -\frac{\partial D(I, I_T)}{\partial b_i} \leqslant 0 \end{cases}$$

where α is constant value that amplifies the output of the gradient and it has an impact on the convergence and the bandwidth overhead. If we pick large value for α, we will take bigger steps toward the target sample and we will add more overhead. We modify the source sample by summing it with Δ, ($I_s^{new} = I_s + \Delta$). We iterate this process, compute Δ for each I_s, and update the source sample until we leave the source class, $f(I_s^{new}) \neq s$ or the number of iterations passes maximum allowed iterations (we set it as 200 iterations).

Because we only increase the bursts where $-\frac{\partial D(I,I_T)}{\partial b_i} > 0$, we may run into the cases that after some iterations $\nabla(-D(I, I_T))$ does not have any positive values or all the positive values are extremely small that they do not make any significant changes to I_s. In such cases, if $I_s^{new} - I_s$ is smaller than a threshold (we used threshold 0.001) for a few iterations (we used 10 iterations) and we are still in the source class, we refill the pool with new samples and pick a new target sample I_T and we continue the process with this new I_T.

5.8 Evaluations

In this section we examine the overhead and undetectability of traces protected by our method. We only evaluate Scenario II, explained in Section 5.6.2, because it was the case that the attacker has high accuracy rate on the Adversarial Traces.

We use traces in Defender Set and generated the defended form of them by the method described in the previous section. We first require a *detector* ($f(x)$) to identify when the generated samples leave their source class. Thus, we define the *detector* a CNN model with the same architecture as Simple CNN, and train it on the traces in Attacker Set. In our evaluations we investigate two cases:

- Case I: We fill the pool with instances from Attacker Set. In this case, the *detector* has been trained on the target classes.

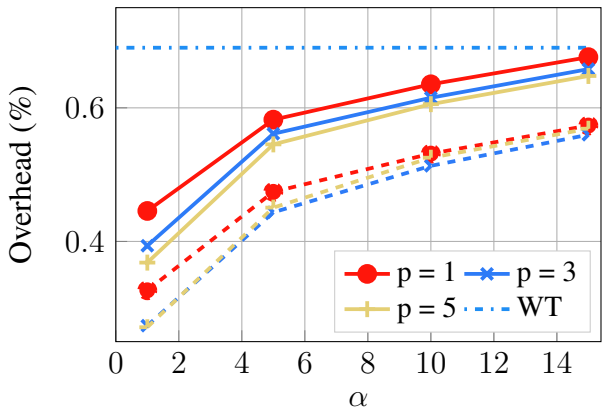
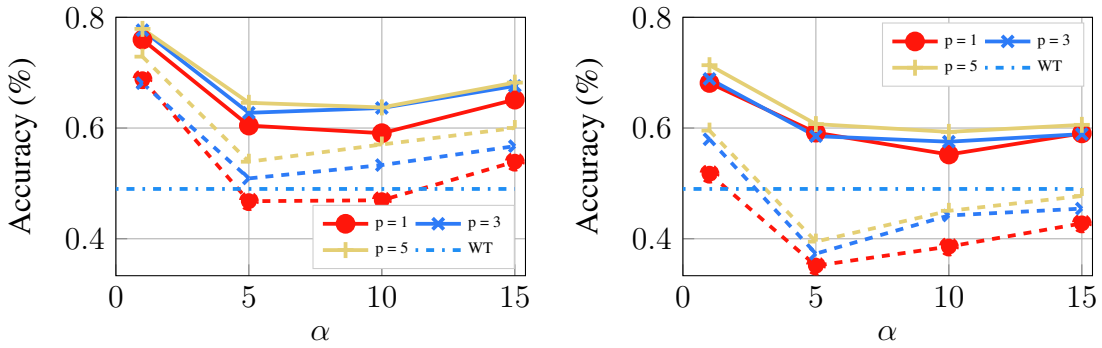


Figure 5.3. **Bandwidth overhead**: this figure shows the bandwidth overhead of generated samples as α and pool size vary. Dashed lines show the results of Case I and solid lines show the results of Case II..

- Case II: We fill the pool with instances from Open-World dataset. In this case, the *detector* has not been trained on the target samples.

We generated defended samples in various settings. We varied α and p to evaluate their effect on the strength of the defended traces and the overhead. We measured the detectability of the defended samples by applying DF attack on them. In DF attack, the default input size to the CNN recommended by Sirinam el al. [72] is 5,000 packets. Because both Walkie-Talkie and our method increase the size of the bursts, the number of packets in the traces increases. Therefore, the first 5,000 packets in the defended traces contain less information compared to the undefended traces. To solve this problem, we pick the 80th percentile packet sequence lengths in the defended traces as the input size of CNN in DF attack. The 80th percentile of packet sequence lengths in the defended traces were around 10,000 packets in both Walkie-Talkie and our defense. Therefore, we set the input size to 10,000.

Figure 5.3 shows the bandwidth overhead in both Walkie-Talkie (WT) and our method for Case I (solid lines) and II (dashed lines) as α and p vary. As shown in the figure, as we increase α the amount of bandwidth overhead increases. The reason for the increase is that



(a) Case I (b) Case II

Figure 5.4. **Accuracy**: this figure shows the accuracy rate of the generated samples against the DF attack. Dashed lines depict the cases when the input size to the DF attack is 5,000 cells and solid lines show the results when it is 10,000 cells..

α controls the size of the steps that we take toward the target samples. the longer steps, the more overhead. Moreover, the pool size does not seem to have significant impact on the bandwidth overhead. However, the smaller pool sizes has a little bit higher overhead in small values of α. The results indicate that Case I leads to lower bandwidth overhead compared to Case II. Therefore, picking target samples from classes that the *detector* has been trained on will drop the overhead. In all the evaluated settings, the bandwidth overhead of our method is lower than Walkie-Talkie's bandwidth overhead. As α = 1 and p = 5 in Case I, the bandwidth overhead of our defense is 27% and it is 60% lower than Walkie-Talkie. On the other end, as α = 15 and p = 5 in Case I, the bandwidth overhead of our defense is 56% and it is 18% lower that Walkie-Talkie's.

Figure 5.4 depicts the accuracy rate of DF attack as α and p vary. The figure shows the accuracy rate of DF attack with 5,000 packets and 10,000 packets input sizes. Figure 5.4(a) and 5.4(b) depict the results of the evaluations in Case I and Case II, respectively. As p increases in both cases, the accuracy rate drops to its minimum and then it slightly starts increasing. On the other hand, increasing α raises the bandwidth overhead monotonically. Changing p does not have significant effect on the accuracy, specially on Case II.

According to Figure 5.4(a), the lowest accuracy rate is 59% as α = 10 and p = 1 and its corresponding bandwidth overhead is 53%. However, in α = 5 and p = 1, the accuracy rate is 60%, almost the same as α = 10 but with bandwidth overhead of 47%. α = 10 and p = 1 in Case II provides the lowest accuracy rate (55%) with bandwidth overhead 63%. In Case II, if we choose α = 5 and p = 1, the bandwidth overhead drops to 58% but the accuracy rate goes up to 59%.

Our evaluations show that Case I provides lower bandwidth overhead than Case II (between 15% to 27% lower) and the detectability of the generated samples is comparable with Case II. It means that it we train the *detector* with target classes, or pick the target samples from the classes that the *detector* trained on, it will causes lower bandwidth overhead. According to our results, the best setting is to pick target samples from the classes that the *detector* trained on and the pool size (p) 1 and α = 5. Walkie-Talkie performs better than our proposed defense mechanism in terms of detectability of the defended traces, and DF accuracy rate is 49% on Walkie-Talkie, which is 18% better than our defense in the best setting (60%), and its bandwidth overhead is 46% higher than our defense (in α = 10 and p = 1 in Case I).

5.9 Conclusions

Website Fingerprinting (WF) attacks are a kind of traffic correlation attacks that do not rely on a powerful adversary. An adversary who has access to the ingress traffic can perform the attack. The accuracy rate of these attacks reach up to 98%. In response to WF attacks, some defenses have been proposed to lower the success rate of the WF attacks. These defenses come at a cost, and they add bandwidth and latency overhead to the traffic. Designing a defense that optimizes the trade-off between the success rate of the attack and bandwidth

latency overhead is challenging. In this chapter we proposed a new WF defense mechanism that offers lower bandwidth overhead with comparable success rate with state-of-the-art WF defense mechanism. We first used the idea of *adversarial examples* in the machine learning to generate Adversarial Traces to cause misclassification in WF attacker. The generated Adversarial Traces are successfully confound the attacker when the attacker is trained on the undefended traces. We showed that Adversarial Traces are highly transferable. We showed that the Adversarial Traces will be identifiable if the attacker is trained on the Adversarial Traces. To solve this problem, we proposed a new mechanism to generate the adversarial traces to confound the adversary even he is trained on the adversarial traces. To protect a traffic trace, we select a target sample (belonging to any class other than the source sample's class), and we add fake packets to the source trace to shorten the distance between the source sample and target sample. Our defense mechanism leads to 47% bandwidth overhead and drops the accuracy rate of the state-of-the-art WF attack from 98% to 60%.

CHAPTER 6

Future work

6.1 Future Work

In this chapter we discuss about the future work that is in line with our work. We propose a new method to ease the problems in the WF defenses.

6.1.1 Website Fingerprinting Defense using GAN

In Chapter 5, we proposed a defense mechanism to defeat the Website Fingerprinting attackers with reasonable cost. One of the disadvantages of the proposed method is that it relies on some target samples to cast the source sample. Which target samples lead to lower overhead?, how many target samples do we require to leave the source class? Do we leave the source class at last? These are the problems with the proposed method.

The goal of the designing a defense is to change the distribution of the data in each site so that they all have almost the same distribution. We want to make the data not to be separable such that no classifiers can classify them. This is the opposite of the goal of Generative Adversarial Networks (GANs).

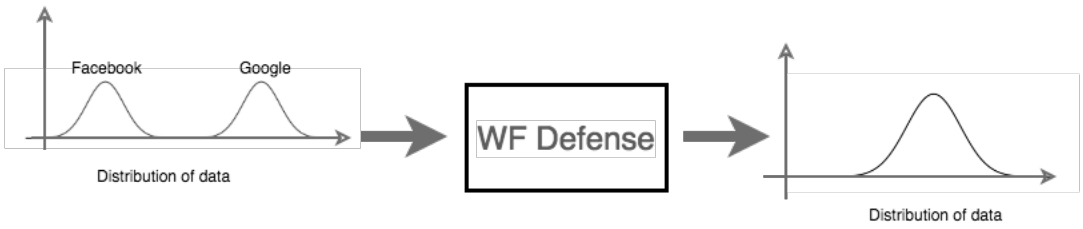


Figure 6.1. WF Defense.

GAN is an unsupervised machine learning technique. GAN generates samples that are superficially authentic and have many characteristics of real samples. GAN is implemented by two neural networks (see Figure 6.2), *Generator* and *Discriminator*. The *Generator* is a neural network that gets a randomized input drawn from a latent distribution and outputs the synthesized samples. The *Generator* should be trained such that the synthesized samples look like to be drawn from the data distribution. The *Generator* acts like a counterfeiter that generates fake notes. The *Discriminator* acts like a police, and its task is to detect whether its input is a synthesized sample or a real sample. The *Discriminator* is a neural network binary classifier. The *Generator*'s objective is to increase the error in the *Discriminator* by causing misclassification in the *Discriminator*. The objective of *Discriminator* is to discriminate the authentic samples from synthesized ones. Two networks compete with each other in *Zero-sum* game framework to reach the equilibrium point. Afterward, the *Generator* learns to map the latent distribution to the data distribution and generate the synthesized samples which look authentic.

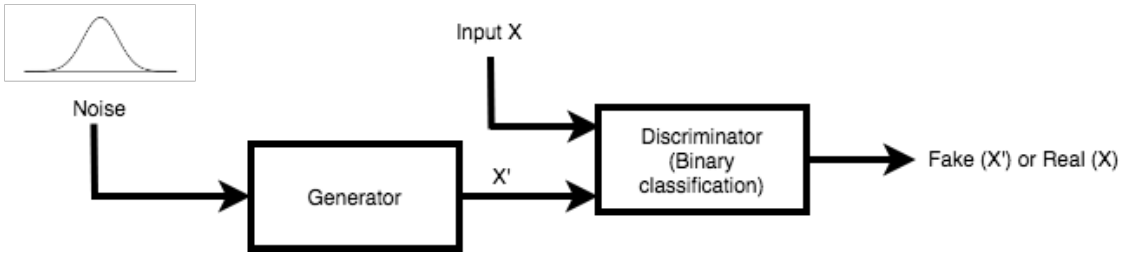


Figure 6.2. GAN architecture.

On the other hand, in designing a WF defense, our goal is to map the data distribution to a latent distribution. The latent distribution can be a random noise distribution, or some target distribution. This is the opposite of the GAN's goal, GAN is mapping the latent distribution to the data distribution. Therefore, we can flip the latent and data distribution's

position in GAN to reach our goal. Figure 6.3 shows the modified architecture of GAN for designing a WF defense.

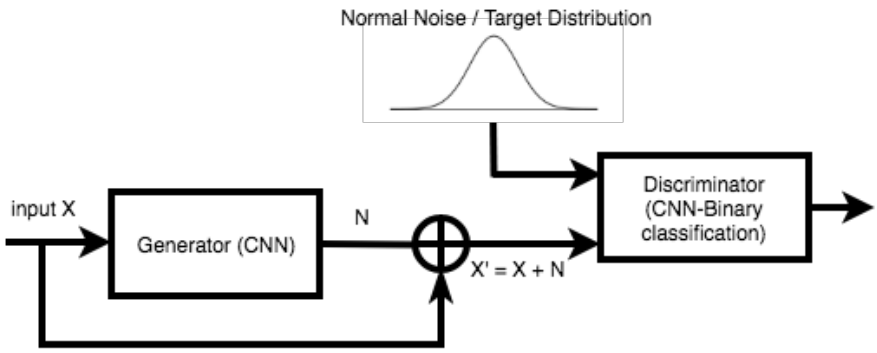


Figure 6.3. WF GAN architecture.

As shown in Figure 6.3, the *Generator* gets the traffic traces (burst sequences),X, as the input and generates the padding (N) that should be added to the burst sequences, depending on the input traces. The *Generator* learns to adjust the padding to mimic the latent distribution. The *Generator*'s objective is to raise the error in the *Discriminator* and keep the amount of padding added to the traffic traces as low as possible to limit the bandwidth overhead. The *Discriminator* is responsible to detect whether its input drawn from the latent distribution or synthesized by the *Generator*. The latent distribution can be a normal distribution. This case is suitable for the un-targeted scenario which the defended traces are all casted to the normal distribution, their distributions are not changed to a particular target distribution. If we want to reshape the data distributions in the websites to the data distribution of a set of target websites, we can use the targets' data distributions as the latent distribution. Therefore, the *Generator* learns to change the data distribution of the sites to the target distribution.

REFERENCES

[1] "AS Organizations Dataset," https://www.caida.org/data/as-organizations/.

[2] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *USENIX Security*, 2004.

[3] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, February 1981.

[4] I2P, "I2P anonymous network," http://www.i2p2.de/index.html, 2003.

[5] R. Dingledine and N. Mathewson, "Anonymity loves company: Usability and the network effect," in *WEIS*, June 2006.

[6] R. Dingledine and S. J. Murdoch, "Performance improvements on tor or, why tor is slow and what were going to do about it," he Tor Project, Tech. Rep., November 2009. [Online]. Available: https://research.torproject.org/techreports/performance-2009-11-09.pdf

[7] B. N. Levine, M. Reiter, C. Wang, and M. Wright, "Timing analysis in low-latency mix systems," in *FC*, Feb. 2004.

[8] A. Johnson, J. Feigenbaum, and P. Syverson, "Preventing active timing attacks in low-latency anonymous communication," in *Proceedings of the 10th Privacy Enhancing Technologies Symposium (PETS 2010)*, July 2010.

[9] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *S&P*, May 2005.

[10] N. Evans, R. Dingledine, and C. Grothoff, "A practical congestion attack on Tor using long paths," in *USENIX Security*, August 2009.

[11] N. Hopper, E. Y. Vasserman, and E. Chan-Tin, "How much anonymity does network latency leak?" *ACM TISSEC*, vol. 13, no. 2, February 2010.

[12] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy traffic analysis of low-latency anonymous communication using throughput-fingerprinting," 2011, available at http://netfiles.uiuc.edu/mittal2/www/throughput-fingerprinting.pdf.

[13] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz, "Denial of service or denial of security?" in *CCS*, 2007.

[14] J. Hayes and G. Danezis, "Guard sets for onion routing," in *Proceedings on Privacy Enhancing Technologies*, 2015.

[15] D. Herrmann, R. Wendolsky, and H. Federrath, "Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier," in *ACM Workshop on Cloud Computing Security*. ACM, 2009, pp. 31–42.

[16] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website Fingerprinting in Onion Routing Based Anonymization Networks," in *ACM Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2011, pp. 103–114.

[17] T. Wang and I. Goldberg, "Comparing Website Fingerprinting Attacks and Defenses," Tech. Rep., 2014, technical report.

[18] T. Wang and I. Goldberg, "Improved Website Fingerprinting on Tor," in *ACM Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2013, pp. 201–212.

[19] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective Attacks and Provable Defenses for Website Fingerprinting," in *USENIX Security Symposium*. USENIX Association, 2014, pp. 143–157.

[20] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in *S&P*, 2012.

[21] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014.

[22] B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi, "Deep text classification can be fooled," 2017.

[23] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," 2015.

[24] N. Carlini and D. A. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *AISec@CCS*, 2017.

[25] J. Geddes, M. Schliep, and N. Hopper, "Anarchy in tor: Performance cost of decentralization," https://arxiv.org/pdf/1606.02385.pdf/, 2017.

[26] A. Acquisti, R. Dingledine, and P. Syverson, "On the economics of anonymity," in *FC*, Jan. 2003.

[27] R. Snader and N. Borisov, "A tune-up for Tor: Improving security and performance in the Tor network," in *NDSS*, February 2008.

[28] T. Wang, K. Bauer, C. Forero, and I. Goldberg, "Congestion-aware path selection for Tor," in *FC*, February 2012.

[29] M. Sherr, M. Blaze, and B. T. Loo, "Scalable link-based relay selection for anonymous routing," in *PETS*, August 2009.

[30] M. Akhoondi, C. Yu, and H. V. Madhyastha, "LASTor: A low-latency AS-aware Tor client," in *IEEE S&P*, 2012.

[31] C. Wacek, H. Tan, K. Bauer, and M. Sherr, "An empirical evaluation of relay selection in Tor," in *NDSS*, February 2013.

[32] R. Jansen, K. Bauer, N. Hopper, and R. Dingledine, "Methodically modeling the tor network," in *CSET 2012*, August 2012.

[33] C. Tang and I. Goldberg, "An improved algorithm for Tor circuit scheduling," in *CCS*, October 2010.

[34] M. Alsabah, K. Bauer, T. Elahi, and I. Goldberg, "The path less travelled: Overcoming tor's bottlenecks with traffic splitting," in *PETS*, July 2013.

[35] R. Dingledine, D. S. Wallach, *et al.*, "Building incentives into Tor," in *FC*. Springer, 2010, pp. 238–256.

[36] R. Jansen, N. Hopper, and Y. Kim, "Recruiting new Tor relays with BRAIDS," in *CCS*, 2010.

[37] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The next-generation onion router," in *USENIX Security*, Aug. 2004.

[38] R. Annessi and M. Schmiedecker, "NavigaTor: Finding faster paths to anonymity," in *Euro S&P*, 2016.

[39] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting," in *CCS*, October 2011.

[40] J. Geddes, R. Jansen, and N. Hopper, "How low can you go: Balancing performance with anonymity in tor," in *PETS*, July 2013.

[41] R. Jansen and N. Hopper, "Shadow: Running Tor in a box for accurate and efficient experimentation," in *NDSS*, February 2012.

[42] "Shadow," http://shadow.github.io/.

[43] J. Qiu and L. Gao, "AS path inference by exploiting known AS paths," in *GLOBECOM*, 2005.

[44] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "RAPTOR: Routing attacks on privacy in Tor," in *USENIX Security*, Aug. 2015.

[45] R. Jansen, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "Never been KIST: Tor's congestion management blossoms with kernel-informed socket transport," in *USENIX Security*, Aug. 2014.

[46] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on tor by realistic adversaries," in *CCS*, 2013.

[47] "CAIDA data," http://www.caida.org/data.

[48] The Tor Project, "Tor metrics portal," http://metrics.torproject.org/.

[49] "Alexa," http://www.alexa.com.

[50] S. Ramachandran, "Web metrics: Size and number of resources," http://code.google.com/speed/articles/web-metrics.html, 2010.

[51] "Maxmind IP geolocation database," http://dev.maxmind.com/geoip/legacy/geolite/.

[52] "Tor users statistics," https://metrics.torproject.org/userstats-relay-table.html.

[53] M. K. Wright, M. Adler, B. N. Levine, and C. Shields, "Passive-logging attacks against anonymous communications systems," *ACM TISSEC*, vol. 11, no. 2, 2008.

[54] L. Overlier and P. Syverson, "Locating hidden servers," in *IEEE S&P*, 2006.

[55] A. Biryukov, I. Pustogarov, and R.-P. Weinmann, "Trawling for Tor hidden services: Detection, measurement, deanonymization," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, May 2013.

[56] R. D. (arma), "Better guard rotation parameters," https://blog.torproject.org/category/tags/guard-relays, Aug. 2011.

[57] R. Dingledine and G. Kadianakis, "One fast guard for life (or 9 months)," in *HotPETs*, 2014.

[58] T. Elahi, K. Bauer, M. AlSabah, R. Dingledine, and I. Goldberg, "Changing of the guards: A framework for understanding and improving entry guard selection in Tor," in *WPES*, 2012.

[59] A. Barton and M. Wright, "Denasa: Destination-naive as-awareness in anonymous communications," in *Proceedings on Privacy Enhancing Technologies*, 2016.

[60] L. Gao, "On inferring autonomous system relationships in the Internet," *ACM/IEEE Transactions on Networks (TON)*, vol. 9, no. 6, 2001.

[61] M. Luckie, B. Huffaker, k. claffy, A. Dhamdhere, and V. Giotsas, "AS relationships, customer cones, and validation," in *Internet Measurement Conference (IMC)*, Oct 2013, pp. 243–256.

[62] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, and kc claffy, "AS relationships: Inference and validation," in *CCR*, 2007.

[63] CAIDA, "The CAIDA AS relationships," January 2016, http://www.caida.org/data/as-relationships/.

[64] N. Mathewson and R. Dingledine, "Practical traffic analysis: Extending and resisting statistical disclosure," in *Proc. Privacy Enhancing Technologies workshop (PET)*, May 2004.

[65] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *IEEE S&P*, 2005.

[66] N. S. Evans, R. Dingledine, and C. Grothoff, "A practical congestion attack on Tor using long paths," in *USENIX Security*, 2009.

[67] M. Edman and P. F. Syverson, "AS-awareness in Tor path selection," in *CCS*, November 2009.

[68] A. Johnson, R. Jansen, A. Jaggard, J. Feigenbaum, and P. Syverson, "Avoiding the man on the wire: Improving tors security with trust-aware path selection," in *24th Symposium on Network and Distributed System Security (NDSS 2017)*.

[69] "Tor directory protocol, version 3," https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt.

[70] Joshua Juen, Aaron Johnson, Anupam Das, Nikita Borisov, and Matthew Caesar, "Defending tor from network adversaries: A case study of network path prediction," in *Proceedings on Privacy Enhancing Technologies*, 2015.

[71] V. Rimmer, D. Preuveneers, M. Juarez, T. V. Goethem, and W. Joosen, "Automated feature extraction for website fingerprinting through deep learning," 2017.

[72] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," 2018.

[73] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis," in *Network & Distributed System Security Symposium (NDSS)*, 2009.

[74] X. Luo, P. Zhou, E. Chan, and W. Lee, "HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows," in *Network & Distributed System Security Symposium (NDSS)*. IEEE Computer Society, 2011.

[75] M. Perry, "Experimental Defense for Website Traffic Fingerprinting," Tor project Blog. "https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting", 2011, (accessed: October 10, 2013).

[76] X. Cai, R. Nithyanand, and R. Johnson, "Glove: A Bespoke Website Fingerprinting Defense," in *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2014, pp. 131–134.

[77] X. Cai, R. Nithyanand, and R. Johnson, "CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense," in *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2014, pp. 121–130.

[78] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses," in *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2014, pp. 227–238.

[79] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a Distance: Website Fingerprinting Attacks and Defenses," in *ACM Conference on Computer and Communications Security (CCS)*, 2012, pp. 605–616.

[80] L. Lu, E. Chang, and M. Chan, "Website Fingerprinting and Identification Using Ordered Feature Sequences," in *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2010, pp. 199–214.

[81] M. Juárez, M. Imani, M. Perry, C. Díaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece,*

September 26-30, 2016, Proceedings, Part I, 2016, pp. 27–46. [Online]. Available: https://doi.org/10.1007/978-3-319-45744-4_2

[82] D. Wagner and B. Schneier, "Analysis of the ssl 3.0 protocol," in *Proceedings of the 2Nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, ser. WOEC'96. Berkeley, CA, USA: USENIX Association, 1996, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267167.1267171

[83] H. Cheng, , H. Cheng, and R. Avnur, "Traffic analysis of ssl encrypted web browsing," 1998.

[84] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009, pp. 31–42.

[85] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale," in *Proceedings of the Network and Distributed Security Symposium - NDSS '16*. Internet Society, February 2016.

[86] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 1187–1203. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes

[87] K. Abe and S. Goto, "Fingerprinting attack on tor anonymity using deep learning," in *in the Asia Pacific Advanced Network (APAN)*, 2016.

[88] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Proceedings of the 25nd Network and Distributed System Security Symposium (NDSS 2018)*. Internet Society, 2018.

[89] R. Nithyanand, X. Cai, and R. Johnson, "Glove: A bespoke website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, ser. WPES '14. New York, NY, USA: ACM, 2014, pp. 131–134. [Online]. Available: http://doi.acm.org/10.1145/2665943.2665950

[90] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in *USENIX Security Symposium*. USENIX Association, 2017, pp. 1375–1390.

[91] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *Proceedings of the 11th European Conference on Research in Computer Security*, ser. ESORICS'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 18–33. [Online]. Available: http://dx.doi.org/10.1007/11863908_2

[92] G. Cherubin, J. Hayes, and M. Juarez, ""Website fingerprinting defenses at the application layer"," in *Privacy Enhancing Technologies Symposium (PETS)*. De Gruyter, 2017, pp. 168–185.

[93] T. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *ICASSP*, 2015.

[94] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," in *Arxiv*, 2016. [Online]. Available: https://arxiv.org/abs/1609.03499

[95] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, pp. 3104–3112. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969033.2969173

[96] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Trans. Audio, Speech*

and Lang. Proc., vol. 20, no. 1, pp. 30–42, Jan. 2012. [Online]. Available: https://doi.org/10.1109/TASL.2011.2134090

[97] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 160–167. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390177

[98] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1701–1708. [Online]. Available: https://doi.org/10.1109/CVPR.2014.220

[99] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[100] D. Mishkin and J. Matas, "All you need is a good init," 2015.

[101] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[102] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," 2015.

[103] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2013.

[104] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[105] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," 2015.

[106] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," 2016.

[107] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," 2017.

[108] L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. S. Schoenholz, and J. Pennington, "Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks," 2018.

[109] R. Overdorf, M. Juarez, G. Acar, R. Greenstadt, and C. Diaz, "How unique is your .onion? an analysis of the fingerprintability of Tor onion services," in *Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS '17)*, November 2017.

[110] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations (ICLR'13)*, 2013.

[111] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, "Circuit fingerprinting attacks: Passive deanonymization of Tor hidden services," in *USENIX Security Symposium*. USENIX Association, 2015, pp. 287–302.

[112] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," 2016.

[113] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," 2015.

[114] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 263–274. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660368